THE ZX80 COMPANION



(SECOND EDITION)

LINSAC

THE MONITOR REVEALED

The ZX80 Companion

(SECOND EDITION)

by

Bob Maunder

Terry Trotter

and Ian Logan

LINSAC

ISBN 0 907211 00 3

LINSAC, 68 Barker Road, Middlesbrough TS5 5ES

© LINSAC 1980

FIRST EDITION:

First printing . . . . . May 1980

Second printing . . . . . June 1980

Third printing . . . . . August 1980

SECOND EDITION:

First printing . . . . . September 1980

All Rights Reserved. No part of this publication may be reproduced or transmitted by any means without the prior permission of LINSAC.
Dr. Logan reserves copyright on Chapter 4.

Printed by Prontaprint, 92 Grange Road, Middlesbrough, Cleveland.

Photographs by Richard and Andrea Nicholson

PREFACE

The Sinclair ZX80 microcomputer has been described as bringing a breath of fresh air to the personal computer scene. Its compactness, price and ease of use have been reflected in the large number of orders placed in the few months following its launch.

''The ZX80 Companion'' arose from the authors' experience in obtaining one of the first ZX80 kits and through many hours of use at home and in the college classroom. It is intended to be a manual for the broad spectrum of ZX80 owners, from the electronics specialist wishing to learn about microcomputers, to the computer professional buying his own system for the first time, and most particularly to the person wanting to use a computer in his own home. Considerable interest has also been shown by educationalists from primary up to tertiary levels. Much of the material in this volume has grown from course notes used in introducing college students to BASIC programming and microcomputers. The ZX80 enthusiast will also find much useful information, particularly in Chapters 3 and 4.

The background of the authors mirrors the variety of backgrounds of ZX80 owners. Terry Trotter graduated from Leeds Polytechnic with a BSc in Electronic Engineering and has several years' experience in designing and using microprocessor-based systems. Bob Maunder has an MSc in Computer Science from Birmingham University and has experience in technical and commercial programming, including CAL techniques, and latterly as a college lecturer. Ian Logan claims to be a typical ZX80 enthusiast: he has no professional connection with computing but is a member of his local hobby computer group, and is a general medical practitioner.

Thanks are due to Clive Sinclair for his permission to use tables of character codes from the ZX80 Operating Manual, to Pat Crabb for two programs used in Chapter 3, and to Margaret Lee for her invaluable assistance with typing.

CONTENTS

Page No.

CHAPTER ONE – OPERATING THE ZX80
- 1.1 Introduction 7
- 1.2 Equipment 7
- 1.3 Loading Programs from Cassette 9
- 1.4 Entering Programs from Listings 10
- 1.5 Running Programs 11

CHAPTER TWO – THEORY OF COMPUTERS
- 2.1 Introduction 13
- 2.2 Elements of the Computer 13
- 2.3 Development of Programs 17

CHAPTER THREE – ZX80 BASIC
- 3.1 Summary 23
- 3.2 Assignments and Calculations 28
- 3.3 Input and Ouput 32
- 3.4 Conditional and Unconditional Jumps and Loops 33
- 3.5 Arrays 37
- 3.6 Character Handling 39
- 3.7 Machine Code 44

CHAPTER FOUR – THE ZX80 MONITOR
- 4.1 Introduction 51
- 4.2 Examining the Monitor 51
- 4.3 Using the Monitor Subroutines 59
- 4.4 The Peripheral Controlling Routines 59
- 4.5 The Print Routines 63
- 4.6 The Arithmetic Subroutines 73
- 4.7 Moving Ramtop Down 77
- 4.8 Moving Display Routine 78

CHAPTER FIVE – CONSTRUCTION AND HARDWARE
- 5.1 Introduction 81
- 5.2 Assembly 81
- 5.3 Hardware 83

CHAPTER SIX – ZX80 PROGRAMS
- E002 Pattern Recognition 91
- U001 Memory Display 96
- E004 Spelling Quiz 97
- E005 Graph Plotter 102
- G005 Hurkle 105
- G016 Four-In-A-Line 108
- G009 The Maze Game 112

APPENDIX ONE – COMPARISON OF ZX80 INSTRUCTION CODES AND Z80 CHARACTER SET

APPENDIX TWO – THE 8K ROM
- A2.1 Introduction 123
- A2.2 New Features 123
- A2.3 Sample Program 126

INTRODUCTION

The manual is arranged so that an intending user of the ZX80 entering computing for the first time may start using the system quickly, and then gradually gain a better appreciation of the machine and its facilities. The informed computer user will find the later sections particularly useful in applying his knowledge to the ZX80.

The first chapter introduces the user to the ZX80 with instructions on how to load and run packaged programs. Chapter two then directs the reader's attention to the basic principles of computers and of programming theory. The largest chapter is the third, being a detailed study of ZX80 BASIC beginning at a simple level and ending with simple machine code programming features. Chapter four is aimed at the enthusiast wanting to know the workings of the ZX80 monitor and thus get the best out of the system. Chapter five guides the owner building his ZX80 from kit form and is also intended to inform users interested in the hardware side of the system. The volume ends with a collection of fully-documented ZX80 programs from the LINSAC catalogue, in Chapter six.

The authors welcome correspondence on any matters raised in the book and may be contacted through LINSAC, 68 Barker Road, Middlesbrough TS5 5ES.

To facilitate study of the manual, portions describing features of the ZX80 more suitable for advanced users are marked with a line in the left-hand margin, as shown. This convention is not followed in Chapter four since the entire chapter is in this category.


sinclair
ZX80
1 2 3 4 5 6 7 8 9 0
NEW LOAD SAVE RUN CONT REM IF INPUT PRINT
Q W E R T Y U I O P
LIST STOP DIM FOR GOTO POKE RAND LET
A S D F G H J K L
CLEAR CLS GOSUB RET NEXT
Z X C V B N M

CHAPTER ONE – OPERATING THE ZX80

1.1 INTRODUCTION

Many ZX80 users will wish to start by using the computer with pre-packaged programs. This chapter details all necessary information for setting up the system and loading and running such programs.

1.2 EQUIPMENT

The ZX80 comes complete with leads for connection to the MIC and EAR sockets of a normal cassette recorder and to the aerial socket of a television set. Figure 1 shows a normal arrangement.

Television

Either a colour or black and white television set may be used, but the latter tends to give a better quality picture. The set should be tuned to channel 36 UHF by means of the normal tuning controls. There is no sound output so the volume should be set to minimum. Connect the single aerial-type lead to the video socket of the ZX80 and to the aerial socket of the televsion. The video output is designed to provide a picture of 24 lines of 32 characters.

Cassette Recorder

A good quality cassette recorder is recommended, preferably one with jack-type MIC and EAR sockets. A tape counter is also very useful when locating programs on cassette. The authors have found a Hitachi TRQ-299 to be very suitable. The twin lead supplied connects the MIC and EAR sockets on the ZX80 to the corresponding sockets on the recorder, by two standard 3.5 mm jack plugs at each end. A recorder with DIN connections may be used but this necessitates obtaining a different lead (from an electrical store such as Currys). Such a lead should have a twin cable and two 3.5 mm jacks at the ZX80 end and a 5 pin DIN plug at the recorder end. It is a good idea to use a lead of about the same length as the one supplied. Overall, jack connections were found to be more reliable than DIN connections.

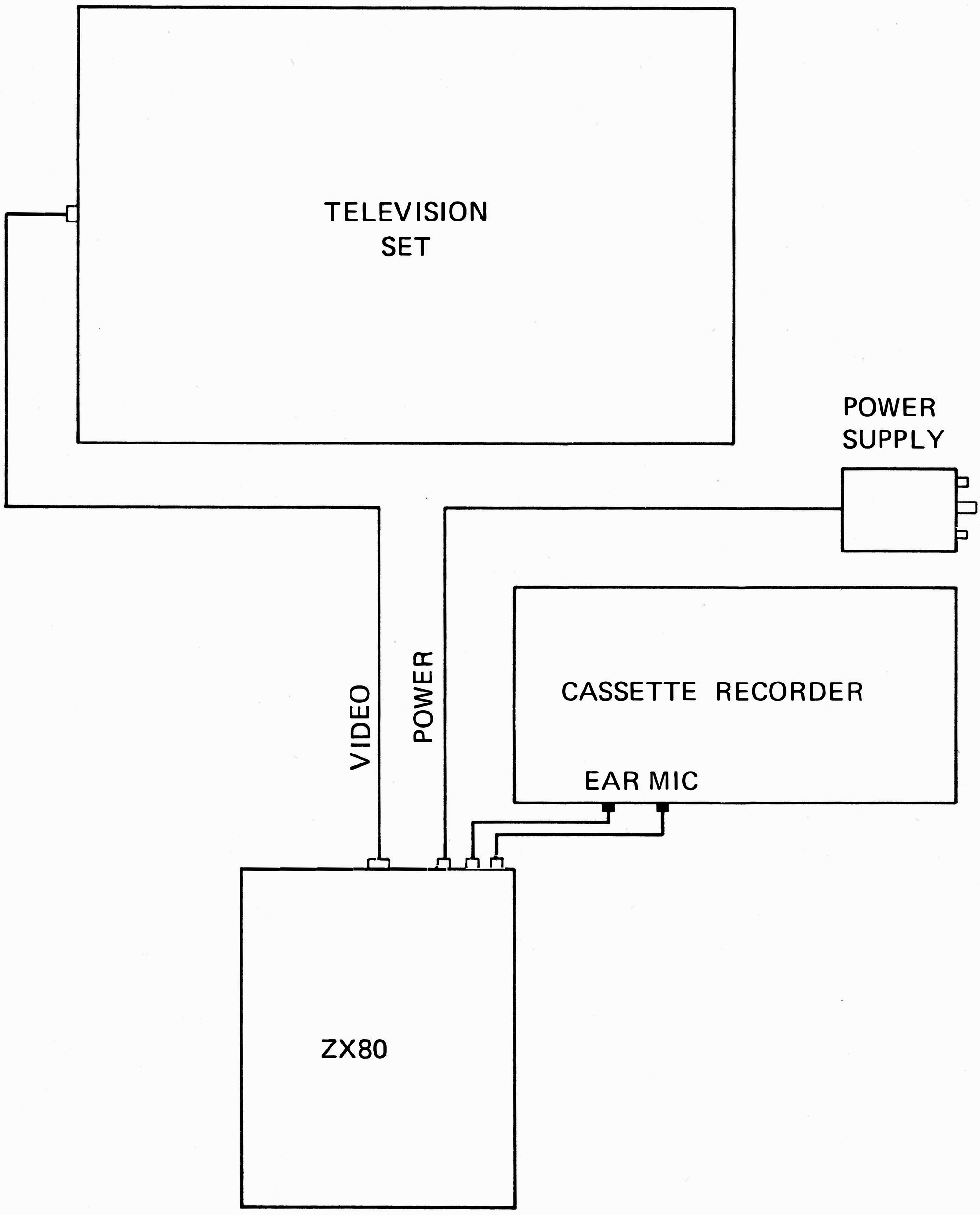


Figure 1 – ZX80 system layout

Power Supply

The power socket of the ZX80 accepts 7—11 V DC with a current of about 400 mA, again connected by a 3.5 mm jack, with positive tip. It is safest to use the supply provided by Sinclair, but providing one's own supply meets the above requirements all should be well. One may find that the thermal fuse in the Sinclair power supply blows if the unit is left on for more than about eight hours.

Switching On

Having received an assembled ZX80, one may check it for the first time as follows: connect the power supply to the POWER socket, and the VIDEO lead as described above and switch on. Tune the television to 36 UHF until the grey dashes on the screen turn to a uniform white and a K (the cursor) is shown at the bottom left-hand corner of the screen. If this does not happen and the television is correctly tuned then either the power supply or the ZX80 itself is faulty. Providing this test is passed the ZX80 may be further checked by loading a program from cassette and running it, as now described.

1.3 LOADING PROGRAMS FROM CASSETTE

Pre-recorded programs for the ZX80 are readily obtainable from LINSAC and are an ideal way for the beginner to learn to use the computer. The following procedure should be followed for loading programs from cassette into the ZX80's memory:

(i) Insert the cassette into the recorder and REWIND to the start.

(ii) With the EAR plug disconnected, press PLAY on the cassette.

(iii) A loud buzz will be heard, followed by a silence and then a high-pitched buzz (the program) — stop the cassette at the beginning of the silence.

(iv) Set volume and tone controls on the recorder to maximum. If there are separate BASS and TREBLE controls, set BASS to minimum and TREBLE to maximum. Connect the EAR jack but not the MIC jack.

(v) Connect the power supply to the ZX80 and with the aerial socket connected a K will be shown as described above.

(vi) Press the W key (load), then the NEWLINE key and immediately press play on the recorder.

(vii) When a number of lines of text are displayed on the screen (after no more than 60 seconds of playing), stop the recorder. The program has now been loaded.

If the program does not load, reset the ZX80 by removing and re-inserting the power lead and repeat the above process. If no success is obtained, the cassette heads may be at fault — clean and, if possible, demagnetise them. A low-quality recorder often gives poor results. Problems may often occur if the ZX80 has become hot — allow it to cool and then try again.

1.4 ENTERING PROGRAMS FROM LISTINGS

ZX80 program listings are even more readily available than cassette software: programs may be found in the ZX80 operating manual, in chapter six of this volume, in hobbyist magazines, or again from the LINSAC catalogue.

To the novice a program listing may look very baffling — it helps however to recognise that a program consists of instructions numbered in the order in which the ZX80 is to follow them. Thus a listing is a collection of numbered lines. Lines are entered by pressing the keyboard keys, and each line is terminated by NEWLINE. However it is not necessary to type in an entire listing letter-by-letter, since certain frequently occurring words (BASIC keywords) have their own keys, e.g. PRINT is entered by pressing the O key, rather than P-R-I-N-T. Such single-key entries may be identified in the listings in Chapter six.

Before starting to enter a new program from a listing press the NEW key (Q) and NEWLINE : this prepares the ZX80 to store a new program. As you enter the first line, what you type will appear at the bottom of the TV screen. When NEWLINE is pressed at the end of the line, the line moves to the top of the screen. As further lines are entered, these too move up the screen. If several lines are entered the initial ones may disappear from the screen: they are still held in the computer however. If at

any stage an S appears, in inverse form, on a line, or if you notice that NEWLINE has no effect, then an error has been made in entering the line. The S shows the position of the mistake: press the RUBOUT (shift O) key repeatedly until the S disappears, and re-enter.

The S also appears if a line is incomplete, and will only disappear when it is completed

1.5 RUNNING PROGRAMS

Pre-packaged software generally comes ~~will~~ WITH full instructions, and these should be followed.

Modes

The ZX80 operates in two modes, command mode and program mode. When a program has been loaded from cassette the program instructions will be shown on the screen. Program mode is entered by pressing RUN, the listing disappears, and the machine remains in this mode while the program is running. If at any point in the running of a program the listing appears on the screen it is clear that the program has terminated for some reason.

Input

The computer works conversationally, that is, in most programs the user enters information from the keyboard and the computer replies by displaying further information on the screen. Note that while the computer is processing, the screen goes blank.

In using a program one may need to enter two types of data, either numbers or text. If the program expects numbers the cursor will appear as an inverse video L, while if text is expected the L will have quotation marks around it. Every input must be terminated by pressing NEWLINE.

If an error is made in entering information (e.g. entry of a number outside of the permitted range – 32768 to 32767) an inverse S will appear on the screen and the NEWLINE key will have no effect. To overcome this, the error must be corrected. This may be done most simply by rubbing out the wrong input and then entering correct information: hold down the shift key and press RUBOUT (number O) repeatedly until the error is removed and then input the data correctly, ending with NEWLINE.

Rerunning

When the program ends, a numerical code will appear at the bottom left-hand corner of the screen, e.g. 0/500. If the number before the / is other than 0 or 9 then the program has ended prematurely with an error: this of course should not happen with a well-produced prepackaged program. To run a program again, first press any key (this causes the ZX80 to revert to command mode, and the program instructions will appear on the screen) then press RUN and NEWLINE, and the program starts again.

Some programs may require the use of the CONT (continue) key after a program has stopped. This simply causes the program to be restarted from where it stopped: often used in two-player quiz programs where one players sets up information for the quiz and then the other player is tested.

Interrupting

A final comment concerns how to stop a program that is running, for example if one realises that inappropriate data has been input. The BREAK key (~~shift and~~ space) may work while the program is being processed (blank screen). Otherwise one may stop the program while it is waiting for input by entering data which will cause a program error. It will become clearer why this works later, but at present follow the following rules.

1. For number input enter 999*99 and NEWLINE.

2. For text input:

   (a) press shift-8 and then RUBOUT twice (This removes the quotes)

   (b) then enter CHR$ (999*99)

In both cases the program terminates with a code 6/.

If one wishes to stop a program and one is unconcerned that it will be lost from the ZX80's memory, then the simplest method is to disconnect and then reconnect the power jack.

CHAPTER TWO – THEORY OF COMPUTERS

2.1 INTRODUCTION

Before launching into programming a computer it is useful to know something of the structure of a computer system and of what is happening when a program is running. This chapter summarizes the main parts of a computer and outlines how computer programs may be developed.

2.2 ELEMENTS OF THE COMPUTER

Summary

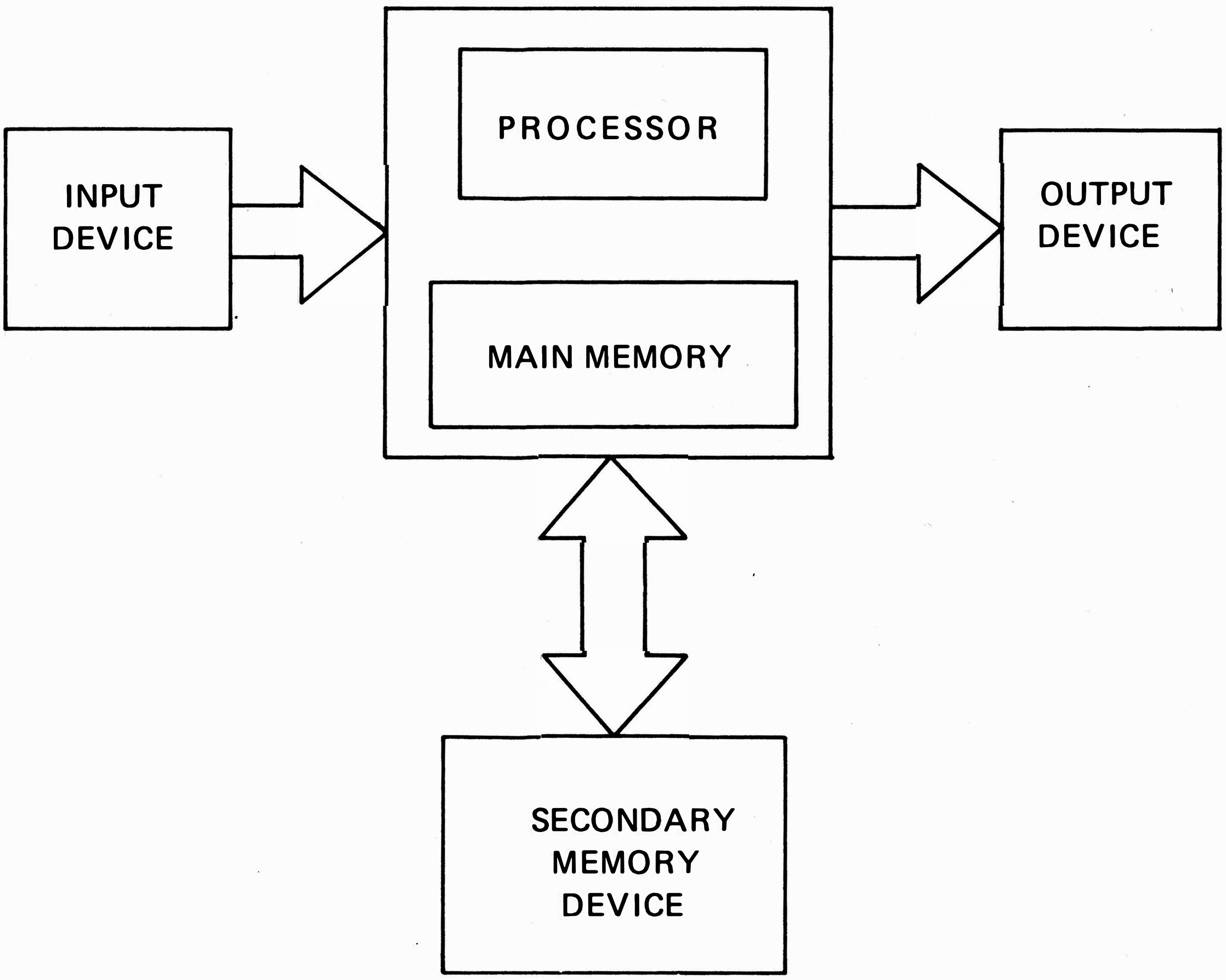


Figure 2 – Computer structure

When we handle information we often do so in three distinct stages — reading, processing and writing. For example, in doing a crossword puzzle we read a clue, work out the answer, and then write it down. This also applies to computer processing of information, and there are devices corresponding to the human facilities enabling us to read, process and write. Information is entered via an input device to the central processing unit (CPU) where it is processed and results sent to an output device. A large part of the CPU is memory, but this may be supplemented by an exterior secondary memory device. These devices appear in the ZX80 as follows:—

ELEMENT	ZX80 FEATURE	FUNCTION
Input device	Touch sensitive keyboard.	Entry of data to CPU.
Output device	TV Screen	Display of results of processing
Processor	780-1 microprocessor	Processing of data
Main memory	ROM and RAM components	Storage of program instructions and data currently in use
Secondary Memory device	Cassette recorder	Storage of programs on long-term basis

These features are now considered in detail.

Input

Each time a key on the keyboard is depressed an appropriate numerical code is stored in an area of main memory.

Output

Results of processing are displayed on the television screen by means of 24 lines of 32 characters each.

Processor

The 780-1 microprocessor can be considered the 'heart', or perhaps more

appropriately the 'brain', of the ZX80 computer. In conjunction with main memory it follows the instructions of a computer program using data supplied. Its three main parts are:

(1) The control unit — controlling the step-by-step execution of a set of program instructions.

(ii) The arithmetic and logic unit — carrying out arithmetic and handling decisions.

(iii) Registers — a very small amount of memory used as the processor's 'scratch-pad'.

Main Memory

Some understanding of memory principles is vital to successful computer programming; this will be considered in more detail.

Main memory at the lowest level consists of groups of two-state devices, e.g. devices which can have high potential or low potential. If one state is represented by the digit 0 and the other by the digit 1 we have a direct link with the binary number system. Such devices in the ZX80's memory are grouped in 8's so that one memory 'slot' or location can hold eight binary digits (0's or 1's). The term binary digit is often abbreviated to 'bit' and a group of eight bits is referred to as a 'byte'.

A given memory location or byte could therefore contain the pattern 0001 0110. To understand what this means, consider a decimal number, e.g. 234 or $(2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0)$, using powers of ten based upon the position of the digits in the number. Similarly in binary 0001 0110 means $(1 \times 2^4) + (1 \times 2^2) + (1 \times 2^1)$ or 16 + 4 + 2 = 22. Thus a byte may contain anything from 0000 0000 to 1111 1111 or 0 to 255.

Numbers are stored in two bytes with the least significant eight bits in the first byte and the most significant eight bits in the second byte, as usual with the Z80 processor.

Positive numbers are stored with a zero in the most significant bit position, thus giving a range of:

0000 0000 0000 0000 to 0111 1111 1111 1111

or 0 to 32767

Negative numbers are held in "two's complement" form i.e., a number -N is stored as (2^{16} −N). This means that negative numbers always have a one in the most significant bit position. To avoid long series of zeros and ones, the hexadecimal number system (base 16) is commonly used as shorthand for binary. This system has digits representing numbers of powers of sixteen and uses the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, the letters representing decimal 10, 11, 12, 13, 14, and 15 respectively.

Thus a hexadecimal number 3E is $3 \times 16^1 + E \times 16^0$
$= (3 \times 16) + (14 \times 1)$
$= 62$

In binary, 62 = 0011 1110
but 0011 = 3
and 1110 = 14 = E hexadecimal

Therefore a binary number may be represented simply in hexadecimal by one hexadecimal digit replacing four binary digits.

Turning to what may be stored in memory, it is appropriate to return to a human analogy. If we wish to work out a problem we need two different types of information, e.g. in calculating the amount of carpet required to cover the floor of a room we need to know:

(i) the dimensions of the room, and

(ii) how to do the calculations, i.e. length x breadth.

Similarly for a computer to solve a problem, it needs to have in its memory both the data required and instructions to follow. Thus the number in a byte may either (partly) represent an item of data or a program instruction.

Secondary Memory

On many computers secondary memory is used to supplement the size of the main memory in the running of a program, in order to increase the amount of data that may be used. At the time of writing such data files on secondary memory are not available on the Sinclair ZX80. However, its other main use is the semi-permanent storage of programs for later reloading. A number of instructions comprising a program may be entered

into main memory via the keyboard. However, when the machine is switched off, memory contents are lost. To avoid having to re-type the program whenever it is required, it may be transferred from main memory onto a cassette by a simple command, and then transferred back again at a later time. The format in which programs are stored on cassette is unique to the ZX80 and is not compatible with that of other computers.

2.3 DEVELOPMENT OF PROGRAMS

Languages

A computer program is a group of instructions written in a programming language. The computer actually obeys instructions in binary machine code which is orientated towards the architecture of the computer. The ZX80 uses the machine code of the Z80 microprocessor. This is discussed further in Chapter Three Section 7. Programming languages nearer to English have been designed to make programming easier, and ZX80 BASIC is an example of such a "high level" language. As in any language, BASIC has grammatical rules which must be followed in order for instructions to make sense.

The ZX80 has an "interpreter" program permanently resident in memory, which translates a BASIC program into binary machine code to be executed by the computer. The art of BASIC programming consists of learning the grammatical rules of BASIC but also, perhaps more important, learning correct methods of programming. These are now discussed by considering the step-by-step development of a single program.

Describing the Problem

Programs are designed to solve problems, either useful problems such as sorting items into sequence, or more leisurely applications such as games. However, it is vital to fully describe the problem before considering how to solve it. Such a description should include details of what is to be input and what is to be output: in fact it is sometimes useful to write down exactly what you expect to appear on the output screen. Consider a simple number guessing game, in which a player attempts to guess a number chosen by the computer. A full description may appear as follows:

Instructions on playing the game are given on the screen. The player then has ten goes at guessing a number between 1 and 100 chosen by the computer. After each guess the player is told whether the guess is too high, too low or correct. If correct the game ends, otherwise the player is invited to guess again. After ten incorrect guesses the number is revealed and the game ends.

Defining the Method

The next stage is to design the method by which the program will solve the problem. Such methods are best learnt initially by studying other people's programs or by reading texts on the theory of algorithms. However, there are certain general considerations which may be followed. It is best to consider the method in separate parts or "modules", each being relatively independent of the rest. The modules may then be developed and tested separately, making life much easier, particularly in detecting any program errors. The modules may be written down in a block chart form or a series of discrete steps. In a large program modules may be themselves broken down into sub-modules. Returning to the number game example we use the following method.

1. INSTRUCTIONS — display instructions

2. CHOOSE NUMBER — choose random number R (1—100)

3. LOOP — 10 times
   - (i) — invite guess and input as G
   - (ii) — COMPARISON — compare G with R and print appropriate message (e.g. too low). If G = R then stop.

4. SHOW NUMBER — display value of R

Note that letters may be used to refer to items of data at this stage. Some programmers like to use "flowcharts" — visual representations of a program method incorporating different shaped boxes linked by flow-lines. Examples may be found in most books on programming. For example, as part of the method detailed above we have:

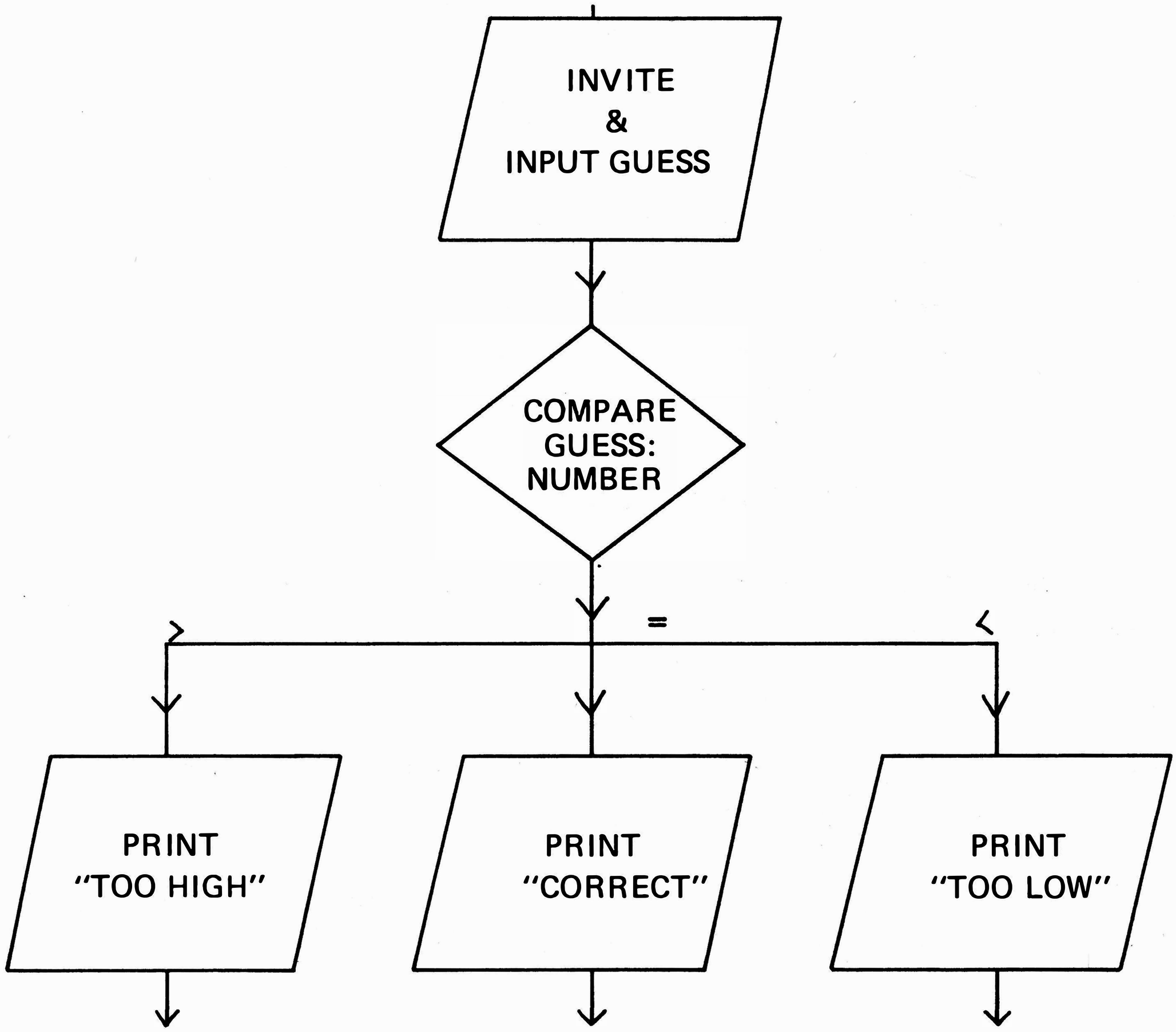


Figure 3 – Flowcharting example

For the ZX80 with only 1K of user memory a large consideration in designing a program method is to the economical use of storage space. This often necessitates keeping printed text, such as instructions, to a minimum. It also means using efficient techniques of coding programs. These are considered in Chapter Three.

Coding the Program

The grammatical rules of BASIC are summarized in Section 3.1. However the number game might be written down and then entered as:

```
10 PRINT, "NUMBER GAME"
20 PRINT  "YOU GUESS A NUMBER"
```

```
30 PRINT "1-100"
40 LET R=RND(100)
 50 FOR I = 1 TO 10
 60 PRINT "GUESS NO. ";I;" ",
 70 INPUT G
 80 IF G<R THEN GO TO 120
 90 IF G>R THEN GO TO 140
100 PRINT G; "IS CORRECT"
110 STOP
120 PRINT G; "IS TOO LOW"
130 GO TO 150
140 PRINT G; "IS TOO HIGH"
150 NEXT I
160 PRINT "HARD LUCK – IT WAS ";R
```

While coding the program make a list of the letters (symbolic names) used to represent data, e.g.

R	=	number chosen at random
I	=	counter showing sequence number of guess
G	=	guess of player

This is to ensure that symbols are not used twice and to provide an aid to understanding the program at a later date.

Testing and Debugging

Often a program, once coded and run, does not run correctly first time and even if it does, this does not necessarily mean it is correct. A program should be tested as thoroughly as possible with varying test data bearing in mind likely users of the program. "Debugging" is the detection of program errors or "bugs" and their correction. Syntax or grammatical errors are normally detected by the system in the entering of program instructions: an S cursor indicator shows this (see Chapter Three).

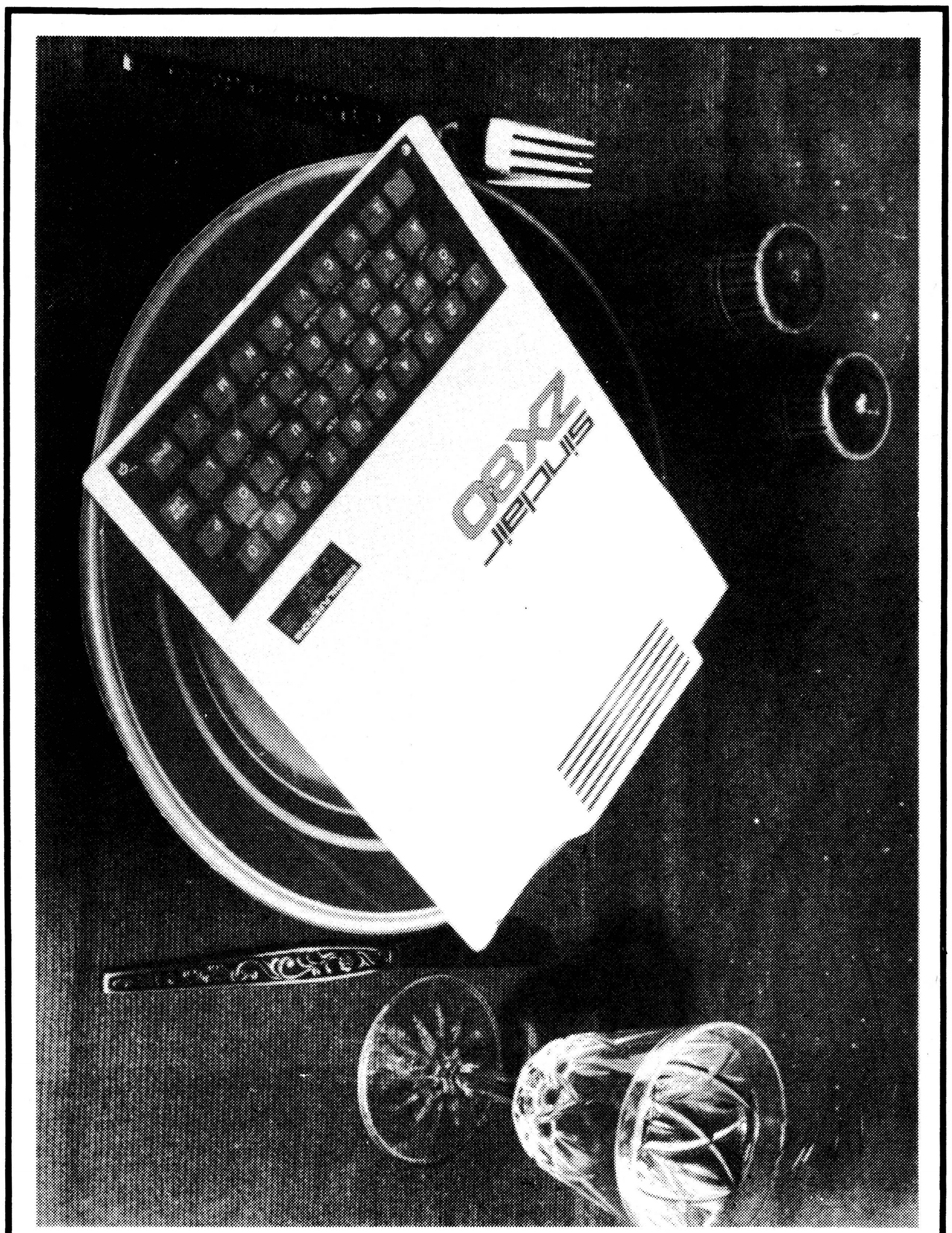
However, errors may also occur in the running of a program and cause it to terminate with an error code. The meaning of this code is determined from the ZX80 Operating Manual and the cause of the error then sought. If difficulty is encountered, extra output statements should be included in the program to both show the flow of the program and also intermediate values of symbols used. Modular programming facilitates

debugging.

Documentation

Having produced a working program, even if only for one's own use, it is very worthwhile to document it. This involves two or three pages of notes — a collection of the notes made in developing the program. Sections to include are description, method, list of variables, program listing, a sample program run and, if someone else will use the program, instructions for use. A documentation file is very useful if one wishes to return to a program some weeks after writing it, or if others wish to use the program: it is amazing how quickly details are forgotten.

Samples of LINSAC documentation appear in Chapter Six.


atari
xe

CHAPTER THREE – ZX80 BASIC

The following summary of ZX80 BASIC is intended for readers familiar with BASIC. Other users should start at section 3.2, having briefly viewed the summary.

3.1 SUMMARY

Lines

ZX80 BASIC has a line-orientated structure, with one program instruction per line. Lines are numbered (maximum of 9999) in the order in which they are to be executed. In command mode the current program listing or part of it is displayed in the upper screen while the instruction being entered is shown at the bottom working area. Intermediate lines may be entered by choosing appropriate intermediate line numbers. A line may be deleted by entering the line number only. A line may be edited by using the cursor control keys (arrows) to move the current line number pointer (inverse >) to the line and then pressing EDIT. This causes the line to appear in the lower working area where it may then be altered, again using cursor control keys, RUBOUT, or insertions as required.

The Display File

The concept of the 'display file' is a crucial one in understanding the workings of ZX80 BASIC. It consists of an area of RAM holding the characters that are 'currently' being shown on the screen. The size of the display file varies according to how much information is on the screen. Its size is limited by the amount of other information in RAM, and in particular the size of the current program and accompanying data. Thus if a large program is entered the display file shrinks: less and less lines are displayed on the screen. Similarly when a program is run, the program may terminate having produced more output than can be held in the display file, although only half of the screen may be taken up. Further details on the display file appear in Chapter 5.

Symbols

Memory locations or variables are referred to by symbolic names, with rules on name formation depending on whether numbers or characters are stored. Numerical names may be of any length composed of letters and numbers but must start with a letter. Character names consist of a letter followed by a dollar sign.

Keywords

ZX80 BASIC is based upon the use of keywords which make up the commands and program instructions used. Most of these may be entered by single key-strokes. A "command" is entered without a line number and is obeyed by the computer directly, whereas a program instruction requires a number and is only obeyed when the program is run.

Cursor

The cursor is used in command mode to indicate whether a keyword (K) or literal (L) is to be entered next. For example if key U is pressed when the cursor is 'K' then IF will be displayed: if however the cursor shows 'L' then simply the letter U will be shown. If an error is made in entering a program line an inverse S will be shown at the point of error and the newline key will not be accepted until the fault has been rectified using editing facilities.

Commands

The following keywords are normal commands but can (not very usefully) be used as program instructions if preceded by a line number. Brackets refer to an optional feature.

LOAD — transfers a complete program (+ data and system variables) from cassette into main memory.

SAVE — reverse of LOAD.

LIST (n) — lists program instructions, either from the start if integer expression n is omitted, or including line n if included, with the current line pointer (>) at this line (useful previous to editing). The amount of

the current program displayed in command mode or in response to LIST depends upon the size of the display file. As stated above, the size of the display file depends upon how much RAM is being used for the current program and variables. If a program uses a large number of variables a larger program listing may be obtained by entering CLEAR (see below) beforehand.

RUN (n) — clears variables and starts program executing at start of line n if specified. GOTO (n) has a similar effect, except that variables are not cleared — enabling a program just loaded to be restarted from its state as stored.

NEW — initialises the system: all program instructions and variables are erased.

CLEAR — erases all variables, but retains current program.

CONTINUE — used to restart a program from where it stopped.

Instructions

The following keywords are program instructions but most may be used without line numbers as direct system commands.

LET name = expression	stores the result of the expression in locations referred to by 'name'.
INPUT name	stores a value input from the keyboard in locations referred to by 'name'.
PRINT list of variables or expressions	displays values on upper screen. Lists may be separated by commas or semi-colons giving different output spacing.
GOTO n	where n is a numerical expression. Causes program execution to continue at line number n or the next line with a number higher than n if line n is not present.

IF condition — THEN instruction	if the 'condition' is not false then the 'instruction' is obeyed. Otherwise no effect. Conditions can contain relational operators >, < and = and connectors AND, OR, NOT.
DIM letter (n)	allocates memory space for an array with a highest subscript given by the expression n. It is referred to by the letter. Only one dimension is allowed.
GOSUB n	Transfers program control to a closed sub-routine at a line number given by numerical expression n. Control is re-turned to the instruction following GOSUB when RETURN is encountered.
RETURN	return from subroutine.
CLS	clears the screen.
REM	a comment statement which may be followed by any symbols: there is no effect on the program.
RANDOMISE n	sets the initial value of the random number generator to n if specified or otherwise to a value based on the TV frame counter.
POKE n1, n2	places the value of numerical expression n2 in memory location n1.

Functions

The following functions may be used in instructions:—

RND (n)	gives a (pseudo—) random number between 1 and n.

ABS (n)	gives the absolute (unsigned) value of expression n.
PEEK (n)	gives the decimal contents of memory location n.
USR (n)	causes entry of a machine code routine at address n and on returning, supplies the value left in the HL register pair, or n if HL is unchanged.

The remaining functions concern character strings.

TL$ (string)	supplies a string identical to its argument without its first character.
CODE (string)	supplies the numerical code of the first character of the string.
CHR$ (n)	supplies a character, the code of which is the value of expression n.
STR$ (n)	supplies a character string equivalent to the value of expression n.

Operators

Expressions may include the operators —

+ (addition), — (subtraction), * (multiplication), / (division),
** (exponentiation), AND, OR, NOT (logical), > (greater than),
< (less than), = (equal)

The experienced BASIC programmer may now study the programs in Chapter Six to see typical uses of ZX80 BASIC and proceed with his own programming. Less informed readers are recommended to study the following more detailed treatment of BASIC, which also includes more advanced features emphasised by a line in the left hand margin.

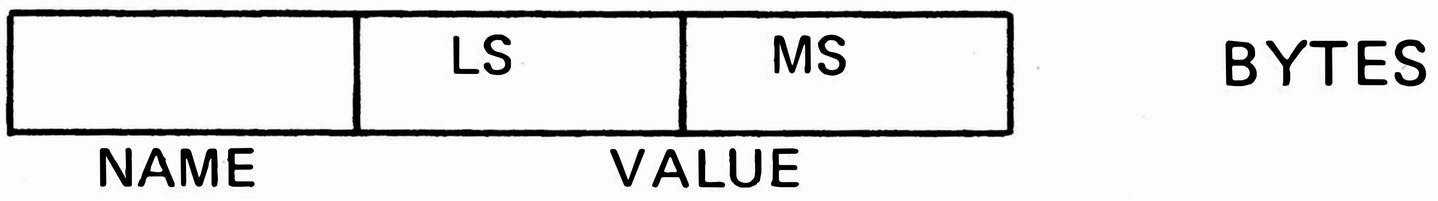
3.2 ASSIGNMENTS AND CALCULATIONS

This section covers the use of the LET statement for numerical expressions and related operators and functions.

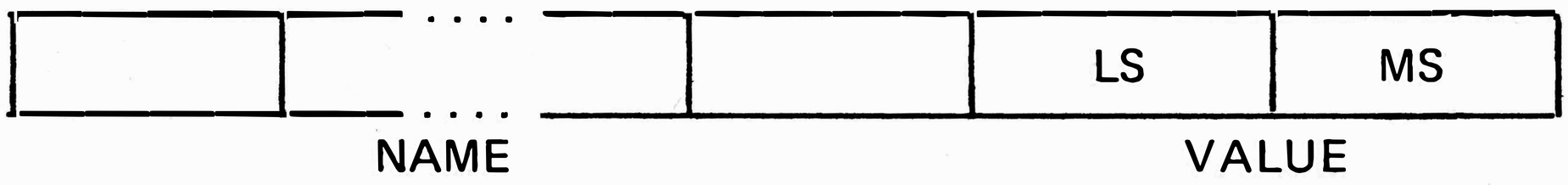
Symbolic Names

As stated in Section 3.1, memory locations containing numbers may be represented by names starting with a letter and consisting of numbers and letters, e.g. V, V1, RESULT, SUB2TOTAL. Such variable names may be of any length, but it is advisable to keep them as short as possible to economise on memory: they are stored as follows:

(i) Single character variable



(ii) Multicharacter variable



N.B. The first byte of the name has its third bit set to zero and the last byte of the name has its first bit set to one, as mark bits.

Expressions

The general form of the LET statement for numerical variables is

line number LET name = numerical expression

A numerical expression consists of variable names or numbers, perhaps linked by operators (see 3.1), similar to a mathematical formula

e.g. LET P = B * C / D – 18

Brackets may also be included

e.g. LET M = (A + B + C) / 3

Operators must be explicit rather than implied. For example the formula

$$y = ax^2 + bx + c$$

would be represented as

LET Y = A * X ** 2 + B * X + C

It is quite valid to have the same variable on both sides of the equal sign. For example to add 1 to the value of K we have

LET K = K + 1

The operators in an expression are evaluated in the following order:

(i) exponentiation (**)
(ii) negation (eg –A)
(iii) multiplication (*)
(iv) division (/)
(v) addition (+) and subtraction (–)

Expressions inside brackets are evaluated first in the order above. The reader may wish to experiment to see the effect of this operator priority. For example the following program may be entered

```
10 LET A = 6 + 30/2 * 3
20 LET B = (6 + 30) / 2 * 3
30 PRINT "A=" ; A
40 PRINT "B=" ; B
```

Logical Values

Relational expressions may also be used in LET statements, since a 'true' value is held in memory as all ones i.e. –1, and a 'false' value as all zeroes, i.e. 0.

For example —

```
LET M = -10 * (A=B)  - 20 * (A>B) + 10 * (A<B)
```

This sets M to 10 if A is equal to B, to 20 if A is greater than B and to −10 if A is less than B.

The logical connectors AND, OR and NOT can be used in a similar way. They obey the rules shown below, taking X as a numerical variable

AND: True AND X gives X
False AND X gives false

OR: True OR X gives true
False OR X gives X

NOT: NOT X gives −X −1

The power of these operators becomes clearer if we consider what happens in memory.

Consider X equal to 2595 and Y equal to 15, i.e.

0 0 0 0	1 0 1 0	0 0 1 0	0 0 1 1	X

0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1	Y

If we have the statement LET A = X AND Y then using the rules above A is

0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1	A

or 3. That is, all but the lowest four bits of X have been masked out. The uses of this may not seem immediately obvious but the following decimal — to — hexadecimal conversion routine illustrates the point. If D is a positive decimal number, it may be converted to a hexadecimal number of four digits, H1, H2, H3, H4 as follows:

```
100 LET H1 = (D AND 28672) / 4096
```

```
110 LET H2 = (D AND 3840) / 256
120 LET H3 = (D AND 240) / 16
130 LET H4 = D AND 15
```

The number could be printed by the following (see Section 3.6)

```
140 PRINT CHR$ (H1 + 28); CHR$ (H2 + 28); CHR$ (H3 + 28);
CHR$ (H4 + 28)
```

The above routine uses "masks" of

```
28672 which is 0111 0000 0000 0000 binary
 3840 which is 0000 1111 0000 0000 binary
  240 which is 0000 0000 1111 0000 binary
and  15 which is 0000 0000 0000 1111 binary
```

and the numbers 4096, 256 and 16 are used to reduce the resulting numbers to within the hexadecimal digit range 0 to 15.

NOTE: Care should be exercised in using relational and logical operators as these are evaluated in priority sequence as are arithmetic operators — see ZX80 Operating Manual, Appendix II.

Functions

There are two numerical functions in ZX80 BASIC, RND and ABS. RND may be used in conjunction with RANDOMISE, which initialises the sequence of random numbers generated by RND.

The general form of the RND function is

RND (numerical expression)

If the value V of the expression is positive then we get a random number between 1 and V; if it is negative we get a random number between V−1 and −32768, or 1 to 32767; if it is zero we get the number 1. Random numbers are useful in many games programs requiring an element of chance. For example to get a random number R representing the code of an alphabetic character (see Section 3.6) i.e. between 38 and 63, we have — LET R = 37 + RND (26) — and the character could be printed by:

PRINT CHR$ (R)

The ABS function gives the positive (modulus) value of an expression.

Integer Arithmetic

The apparent disadvantage of ZX80 BASIC that only whole numbers are allowed can be useful in some circumstances. Any expressions that would normally evaluate to a number involving a fractional part are truncated to the next lowest integer.

For example — LET N = 13/5 — gives N equalling 2.

Thus to find whether a number A is divisible by another number, say 2, we may have, e.g.

IF A = (A/2) * 2 THEN PRINT "EVEN"

3.3 INPUT AND OUTPUT

The INPUT Instruction

There is only one instruction for entering data into ZX80 programs, the INPUT statement. Its form is

line number INPUT variable name

Its effect is to print the cursor at the start of a line on the screen, either enclosed by quotes if characters are required, or alone if a number is required. Data entered in response to INPUT is not automatically printed on the screen.

The PRINT Instruction

The output statement has the form

line number PRINT list of variables, numbers or text.

The list elements may be separated by commas or semicolons, the former causing alignment at the 9th, 17th or 25th positions on a line,

and the latter causing elements to be printed immediately after each other.

e.g. PRINT 1, "VALUE OF B=";A * 6

assuming A has a value of 12 produces

1 VALUE OF B=72

Anything inside inverted commas is printed exactly, whereas items outside are evaluated and then printed. Each PRINT statement produces output on a new line, unless the previous output statement ended with a comma or semicolon.

e.g.
```
10 PRINT "MAXIMUM VALUE =";
20 INPUT M
30 PRINT M
```

If the value 26 is entered when prompted, this produces

MAXIMUM VALUE = 26

PRINT may also be used as a command i.e. without a line number. This is useful when a program has terminated abnormally, since values of variables at the point of termination may be printed out.

It is sometimes helpful to print out information in inverse form (white on black) to highlight certain items. Methods of doing this are outlined in Section 3.6.

3.4 CONDITIONAL AND UNCONDITIONAL JUMPS AND LOOPS

Routines

Program instructions are normally obeyed in line number sequence. However sometimes transfer of control to another part of the program is required, particularly if the program has been written in modules (see Chapter 2). Transfer to an open routine is achieved by the GO TO statement while transfer to a closed routine requiring a return to the original position afterwards is done using the GO SUB statement. A closed routine or sub-routine is used when a module is to be used several times at different points in the program. It terminates with a RETURN

statement which sends control back to the statement following the original GO SUB. The transfer of control may be made conditional through the use of the IF statement.

The GO TO Instruction

Consider first the GO TO statement. Its general form is

line number GO TO numerical expression

and it has the effect of sending control to the line with a value equal to or (if no such line exists) greater than that of the numerical expression. Execution then continues normally from that line. It is known as an unconditional jump. Simple example:

```
10 PRINT "HELLO"
20 GO TO 10
```

This prints the word HELLO successively on the screen until it is full and an error occurs.

More complete example:

```
10 PRINT "ENTER 1–4"
20 INPUT N
30 GO TO N * 100
100 PRINT "GENESIS"
110  STOP
200 PRINT "EXODUS"
210 STOP
300 PRINT "LEVITICUS"
310 STOP
400 PRINT "NUMBERS"
```

This prints GENESIS if 1 is entered, EXODUS if 2 is entered and so on up to 4. When using such expressions in the GO TO statement, care should be taken to avoid out of range values: this may be done using the IF statement as will be seen later.

GO TO may be used as a command (without a line number) to start off

a program at a point while preserving values of stored variables. This is particularly handy after LOADing a program from cassette since values of program and system variables are stored on cassette as well. In fact, once a program has been developed space may be saved by setting up all initial data values (by INPUT or LET instructions) at the start of the program, then stopping, erasing the appropriate INPUT or LET statements and then storing the program on cassette (using SAVE). When required the program is LOADed, and restarted by the GO TO command.

The GO SUB instruction has a similar form to GO TO:

line number GO SUB numerical expression

and has identical effect except that the computer also stores the number of the line following it, so that it may be returned to afterwards. It is known as an unconditional sub-routine jump. The single word instruction RETURN terminates the routine jumped to.

Example of the use of these statements:

```
    • • • •
150 GOSUB 500
160 • • •
    • • • •
500 PRINT "PRESS NEWLINE TO CLEAR SCREEN"
510 INPUT Z$
520 CLS
530 RETURN
```

The instruction at line 150 invokes lines 500-530 and then control is returned to line 160.

The IF Instruction

The conditional IF instruction has the form:

line number IF condition THEN instruction

and the instruction part is executed unless the condition is false (or 0, see Section 3.2). The condition part may be made up of the conditional

operators and conjunctions mentioned previously, i.e.

>,<, =, AND, OR, NOT

and the instruction part may be any valid instruction, including another IF statement: the most commonly used instruction is GO TO. For example a simple input checking statement is shown below at line 30.

```
10 PRINT "ENTER 1–4"
20 INPUT N
30 IF N<1 OR N>4 THEN GO TO 20
40 PRINT N
•  •  •  •
```

A short program to find the largest of list of numbers (ending with zero) uses the IF statement as shown:

```
10 LET M = –999
20 PRINT "ENTER NUMBER OR 0 TO STOP"
30 INPUT N
40 IF N = 0 THEN GO TO 70
50 IF N>M THEN LET M = N
60 GO TO 20
70 PRINT "LARGEST NUMBER WAS ";M
```

Use of the IF statement may be extended to repeat a section of a program a number of times (a loop). The example prints a table of squares of numbers up to ten.

```
10 PRINT , "NUMBER", "SQUARE"
20 LET N = 1
30 PRINT ,N,N * N
40 LET N = N + 1
50 IF N>10 THEN STOP
60 GO TO 30
```

Here N is being used as a counter as well as data.

The FOR Instruction

Loops may be implemented in a simpler way using two statements in conjunction — the FOR and NEXT statements. These have the form:

line number FOR letter = start expression TO end expression

• • • (section of code to be repeated)

line number NEXT letter

For example

```
10 FOR C = 1 TO 10
20 PRINT "MESSAGE"
30 NEXT C
```

displays MESSAGE ten times. Thus the above NEXT statement is equivalent to

```
30 LET C = C + 1
40 IF NOT C > 10 THEN GO TO 20
```

Thus at the end of the loop the counter has a value one greater than the end value specified in the FOR statement. Note also that the counter is incremented by one each time — if a different increment is required then LET and IF statements must be used to specify the loop.

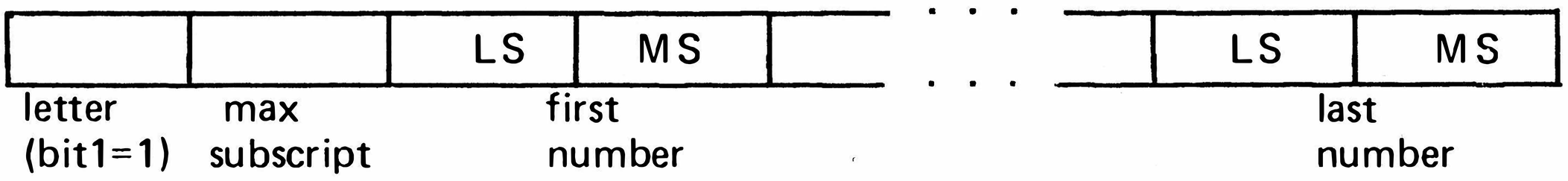
Further uses of loops appear in the following section.

3.5 ARRAYS

Definition

An array is a group of numbers referred to by a single letter, with individual numbers in the group identified by numbered subscripts. For example, if A refers to the numbers 1, 6, 4, 2 then A (0) is 1, A (1) is 6, A (2) is 4 and A (3) is 2.

It is stored in memory thus



The DIM Instruction

Before an array may be used in a program its size or dimension must be defined using the DIM statement, of the form

line number DIM letter (highest subscript)

Subscripts are numbered from zero, so an array DIM A (10) has a maximum of eleven numbers in it. It is permissible to name an array by a letter identical to that used for another variable in a program, each remaining separate through the different storage formats.

Arrays are often used in loops. The following program sets up an array of numbers, then puts them into numerical order using a simple sorting algorithm, and finally prints out the sequenced array.

```
10 DIM A (9)
20 FOR I=0 TO 9
30 PRINT "NUMBER";I;" = ";
40 INPUT A(I)
50 PRIN TA(I)
60 NEXT I
70 FOR I = 0 TO 8
80 LET M = –999
90 FOR J = I TO 9
100 IF A (J)<M THEN GO TO 130
110 LET M = A(J)
120 LET X = J
130 NEXT J
140 LET T = A(I)
150 LET A(I) = A(X)
160 LET A(X) = T
170 NEXT I
180 PRINT "SORTED LIST"
190 FOR I = 0 TO 9
200 PRINT A(I);" ";
210 NEXT I
```

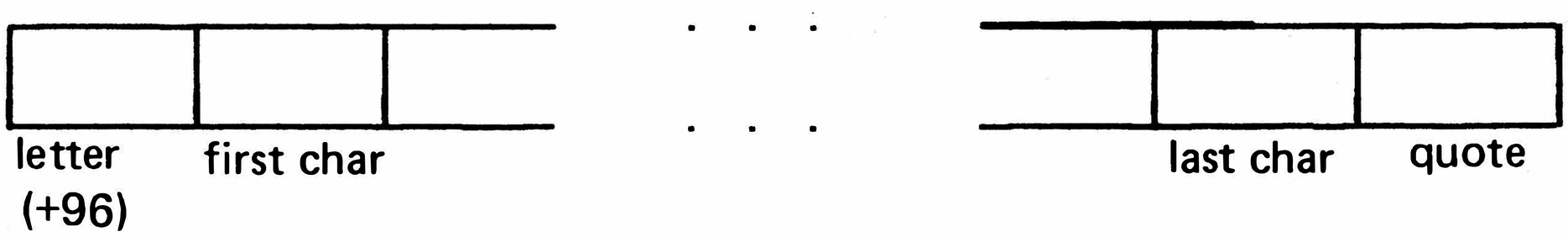
Arrays are often utilised in conjunction with character codes as is now discussed.

3.6 CHARACTER HANDLING

Storage

Thus far only numerical data has been considered, but many applications require the manipulation of letters or other symbols or even letters and numbers together. Any data that is not to be used mathematically may be stored in character format, that is, by means of a unique numerical code for each character. Groups of characters (i.e. numbers, letters or special symbols) are often referred to as strings.

Character data is referred to by variable names of the form of a letter followed by a dollar sign, e.g. A$, C$, X$, and is stored in memory as shown below.



The numerical codes representing characters are shown in Appendix 1. For example the letter A is represented by the binary pattern 0010 0110, i.e. decimal 38. Special picture or graphics characters can also be stored, and even inverse characters i.e. white characters on a black background. The latter do not appear on the keyboard but are reasonably accessible as will be shown later. Note that the code for an inverse character is 128 more than its normal equivalent — it has the first bit set to one. Because of the ZX80's unique keyword system there are also codes standing for complete BASIC keywords, as shown in Appendix 1.

It should be noted that although a number may be stored in a string variable it is still impossible to do arithmetic with the variable. For example

```
10 LET A$ = "20"
20 PRINT A$ *5
```

is incorrect, since 20 is stored as characters rather than in pure binary form.

As shown above, string variables may be assigned values using the LET statement, with the data inside quotation marks. As an alternative, INPUT instructions may be used and these cause the cursor inviting input to appear inside quotation marks. If these quotes are inadvertently rubbed out using cursor control characters the input will not be accepted. Programs may be personalised through the use of string variables representing the user's name as follows:

```
10 PRINT ''WHAT IS YOUR NAME?''
20 INPUT N$
30 PRINT "HOW ARE YOU, ";N$;"?"
```

String manipulation is not limited to simple assignments or output. Relational operators (> and <) may be used with characters in the same way as with numbers, except that they refer to alphabetic order, or rather order of codes (effectively the same).

```
10 PRINT ''WORD1 = '';
20 INPUT A$
30 PRINT A$;'' WORD2 ='';
40 INPUT B$
50 PRINT B$
60 IF A$<BS THEN PRINT A$ ;'' COMES BEFORE ''; B$
70 IF A$>B$ THEN PRINT B$;'' COMES BEFORE ''; A$
80 IF A$ = B$ THEN PRINT "THE SAME"
```

(Not good style!)

Any code may be entered in the above program, in fact any combinations of symbols may be tried. The way in which the character codes are designed means that MAN is evaluated as less (or earlier in alphabetic sequence) than MANKIND. Unfortunately this seemingly powerful facility is limited by the fact that arrays of strings are not available in ZX80 BASIC — otherwise word sorting routines could be used in programs.

Functions

The full power of string handling is realised with the addition of string functions. ZX80 string functions are somewhat unusual but permit

most operations to be carried out.

CHR$ (numerical expression) gives the character whose code is the value of the numerical expression.

CODE (string or string name) gives the code of the first character of the string.

TL$ (string or string name) supplies a string identical to the one in brackets minus its first character.

STR$ (numerical expression) supplies the value of the expression in character form: up to five digits and leading minus sign if negative.

An important concept in character handling is that of the empty or nul! string e.g. TL$ ("A"). If it was required to test Z$ to see whether it was empty we could use —

```
IF Z$ = ""THEN PRINT "EMPTY"
```

or

```
IF CODE (Z$) = 1 THEN PRINT "EMPTY"
```

although character code 1 is quotation mark it can also be used to test for the null string because of the manner in which character variables are stored in memory.

The following program illustrates the use of null strings:

```
10 LET A$ = "ANY STRING"
20 GO SUB 500
30 PRINT
40 STOP
500 IF A$ = ""THEN RETURN
510 PRINT CHR$ (CODE (A$) + 128);
520 LET A$ = TL$ (A$)
530 GO TO 500
```

The subroutine at line 500 onwards considers A$ one character at a time and prints it in inverse video, using the fact that an inverse character has a code of 128 more than its normal equivalent. The TL$ function is

used to gradually cut down A$ until it is empty, at which point the routine exits.

Inverse video can be helpful in emphasizing certain outputs in a program for example titles or errors. A technique is used in the spelling quiz program in Chapter Six to highlight where the student has spelt a word incorrectly. Another way of printing inverse characters without the necessity of a separate routine is shown in Section 3.7.

Character strings often need to be held as separate character codes and arrays may be used for this purpose. The following program inputs a word, stores its component character codes in an array, and then prints it in reverse order. There is a limit of thirty characters.

```
10 PRINT "ENTER WORD"
20 INPUT W$
30 PRINT W$
40 DIM C(29)
50 FOR I = 0 TO 29
60 LET C(I) = CODE (W$)
70 LET W$ = TL$ (W$)
80 IF W$ = "" THEN GO TO 100
90 NEXT I
100 LET I = 29
120 PRINT CHR$ (C(I));
130 LET I = I-1
140 IF I>-1 THEN GO TO 120
```

Characters are displayed through the character generator at the end of the monitor (discussed further in Chapters Four and Five). The following program by Pat Crabb shows how a word may be displayed in large characters on the screen, in 4-character groups.

```
100 DIM X (3)
110 DIM T (7)
120 GO SUB 1000
130 LET A$ = "I AM THE ZX80"
140 GO SUB 700
```

```
150 GO SUB 800
160 INPUT B$
170 CLS
180 GO TO 140
700 FOR I = 0 TO 3
710 LET X (I) = CODE (A$)
720 LET A$ = TL$ (A$)
730 NEXT I
740 RETURN
800 FOR N = 0 TO 7
810 FOR I = 0 TO 3
820 LET Q = PEEK (3584 + 8 * X (I) + N)
830 GO SUB 900
840 NEXT I
850 NEXT N
860 RETURN
900 FOR M = 0 TO 7
910 LET G = 1 + (Q<T(7-M))
920 LET Q = Q - G * T (7-M)
930 PRINT CHR$ (128 *G);
940 NEXT M
950 RETURN
1000 FOR M = 0 TO 7
1010 LET T(M) = 2**M
1020 NEXT M
1030 RETURN
```

The 6 x 8 matrices of alphanumeric characters held in ROM location 3584 onwards are displayed in enlarged form by testing the bit patterns (see Chapter Five). Character 128 (inverse blank) is used to display the character, but other possibilities which may be substituted for 128 at line 930 are X(I) or (X(I) + 128) – these produce interesting effects.

3.7 MACHINE CODE

Introduction

Once the user has mastered BASIC there are still many areas of the system which may be explored using the functions and instructions providing direct access to memory locations and machine language. As stated previously machine language or machine code is the language directly executed by the ZX80 processor — it does not require prior translation as does BASIC. Appendix 1 gives a list of the possible contents of a memory byte (0-255) together with the Z80 language operation which it represents. The instruction is written in a mnemonic form (e.g. LD means load) known as assembly language. The list covers only single byte instruction codes and is therefore not entirely comprehensive.

The operations discussed in this section are orientated towards the ZX80's internal design and particularly the architecture of its microprocessor. It is recommended that users intending to enter this area of programming should obtain information on the Z80 microprocessor family: Rodnay Zacs "Programming the Z80" is particularly recommended.

Instructions and Functions

Two ZX80 facilities exist to access bytes directly. The PEEK function followed by an expression representing an address in brackets gives the decimal contents of the specified location, e.g.

```
LET C = PEEK (1024)
```

sets C to the decimal contents of byte 1024. Conversely the POKE instruction puts data into a memory location and has a form of line number POKE exp1, exp2 — the first expression specifying the address and the second being the value to be inserted,e.g.

```
30 POKE 16384,6
```

places 6 into byte 16384.

The most powerful ZX80 low-level facility is the USR function. It is followed by a numerical expression in brackets and this expression

provides an address value to which control is transferred. The resulting value of the USR function is whatever the routine leaves in the register pair HL, or if HL is unchanged the original address value is given, e.g.

```
20 LET X = USR (16427)
```

with location 16427 containing hex. 21
16428 " " 00 (LD HL, 0100H)
16429 " " 01
16430 " " C9 (RET)

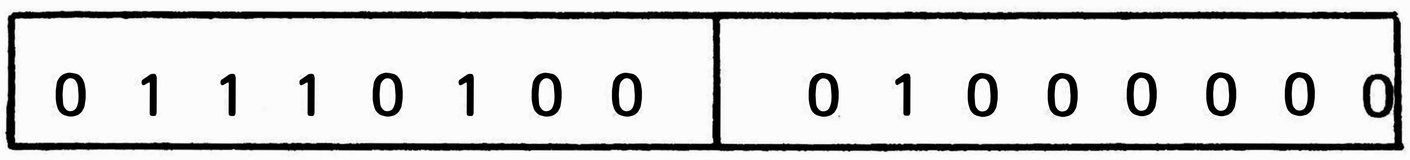
causes the value 256 (0100 hexadecimal) to be placed in X.

Typical uses of PEEK, POKE and USR are now considered.

Systems Variables

The monitor and BASIC interpreter of the ZX80 is held in read-only memory (ROM) but requires a certain amount of working space in random access memory (RAM). Such "system variables" held in this area may often usefully be accessed by programs — details may be found in Appendix III of the ZX80 Operating Manual. Figure 4 on the next page shows some of these variables.

Most systems variables, in the same way as BASIC numerical values, are stored in a pair of consecutive bytes, the least significant byte first. As an example the pointer to the start of the display file, held in bytes 16396 and 16397, may contain the number 16500. This is held as



To obtain this value for use in a BASIC program the following instruction may be used — it puts the value into a variable C.

```
LET C=PEEK(16396)+256*PEEK(16397)
```

Some system variables (e.g. the seed for the random number generator) may be negative so, taking addresses 16412 and 16413 as examples, we require

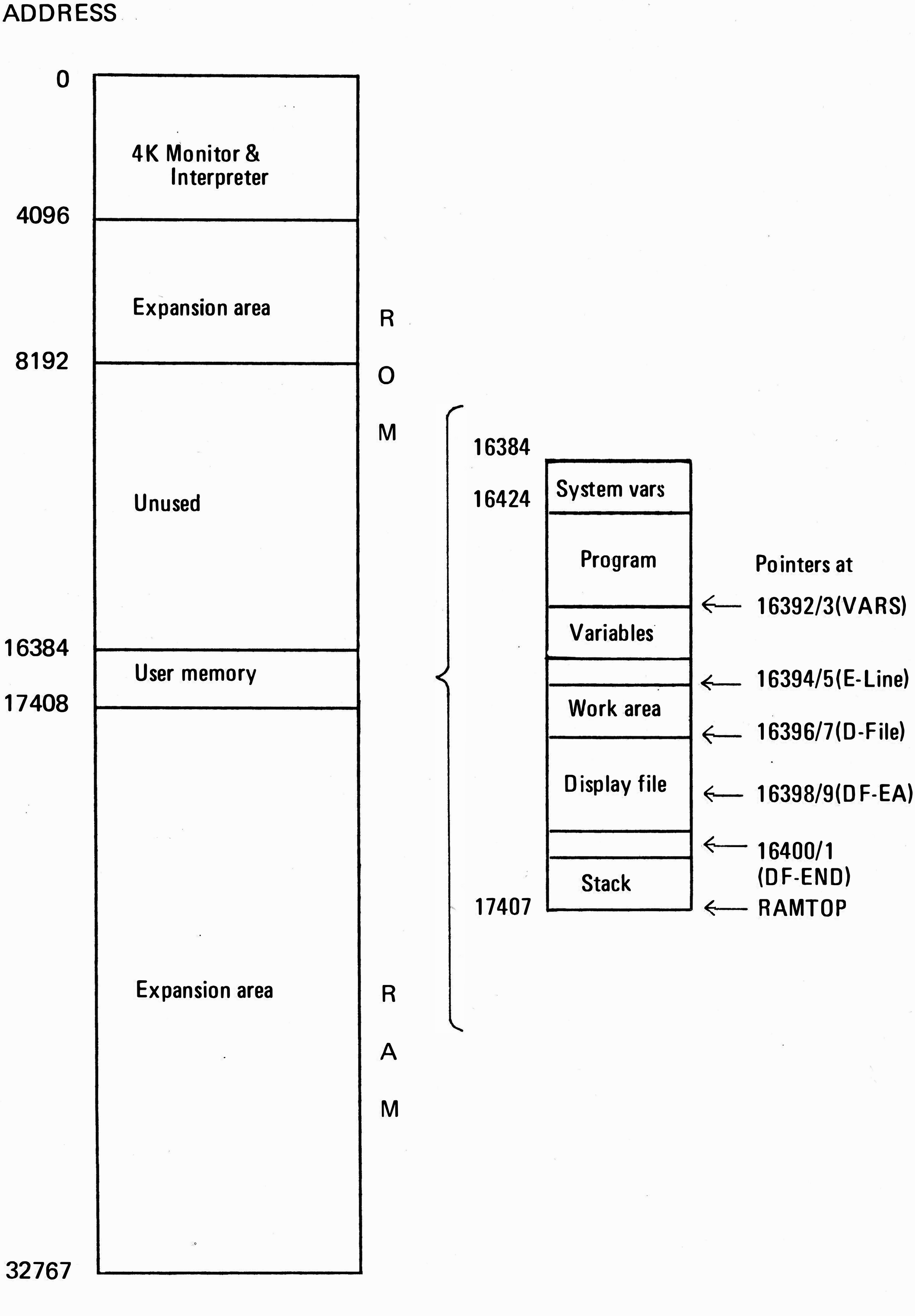


Figure 4 – ZX80 Memory layout

```
10 LET M=PEEK(16413)
20 LET C=256*(M+256*(M>127)+PEEK(16412)
```

The POKE instruction is used in an analogous manner.

Inverse Printing

In considering the use of some of the system facilities, we look first at the program area (see Figure 4) starting at address 16424. The following routine may be used to convert text in PRINT statements to inverse video.

```
10 PRINT "  ANY TEXT  "
•••••••
499 STOP
500 LET Q=0
510 FOR A=16424 TO 16499
520 LET C=PEEK(A)
530 IF C=1 AND Q THEN STOP
540 IF Q THEN POKE A,C+128
550 IF C=1 THEN LET Q=-1
560 NEXT A
```

Lines 10 to 499 represent the main program and the routine at line 500 does the inversion. The program should be first run from line 500 (by RUN 500) and the listing will then reveal the text in line 10 as being in inverse video. Lines 500 onwards may then be erased and the main program SAVEd normally on cassette. The routine may easily be modified to cover more PRINT statements.

Timing

Many games programs require a time limit to be placed on input, i.e. data will only be accepted if entered within a certain number of seconds. This may be achieved using the TV frame counter held in bytes 16414 and 16415. This value is incremented fifty times each second and it is therefore simple to zeroise it before input and then PEEK its contents immediately afterwards. An example is given on the next page.

```
100 PRINT "ENTER DATA"
110 POKE 16414,0
120 POKE 16415,0
130 INPUT D$
140 IF (PEEK(16414)+256*PEEK(16415)-4)>T*50 THEN
PRINT "TOO SLOW"
```

The 4 is subtracted at line 140 to allow for the 80 msecs for the input data to reach the processor. T represents the number of seconds allowed for input.

Machine Language Routines

As stated earlier this book does not seek to teach Z80 machine language. However for users with such knowledge the following section shows how machine code routines can be set up and used from BASIC programs.

The following program by Pat Crabb (with line 235 added by the authors) may be used to enter a routine. Note that the REM statement at the beginning of the program is used to hold the routine.

```
100 REM [□□□□□□□□ . . .□□□]   ( 64 shift A's)
110 LET S=16430
120 LET A=S
130 INPUT A$
140 PRINT A,A$
150 LET H=CODE(A$)-28
160 LET A$=TL$(A$)
170 LET L=CODE(A$)-28
180 LET D=16*H+L
190 POKE A,D
200 IF D=201 THEN GO TO 230
210 LET A=A+1
220 GO TO 130
230 INPUT A$
```

```
235 IF A$="Z" THEN STOP
240 LET V=USR(S)
250 LET A=(V AND 15)+28
260 LET B=(V AND 15*16)/16+28
270 LET C=(V AND 15*256)/256+28
280 LET D=(V AND 7*4096)/4096
290 LET D=D-8*(V<0)+28
300 PRINT CHR$(D);CHR$(C);CHR$(B);CHR$(A)
```

Machine code instructions should be entered in hexadecimal, one byte at a time and terminating with C9 (the Z80 RET instruction). If newline is then entered the routine executes. If however the routine is to be used in a different BASIC program, Z may be input instead of only newline and the program will stop. All lines but the REM statement may then be deleted and the supporting BASIC program can be added and used. The REM statement will contain characters whose codes are equivalent to Z80 operation codes – Appendix 1 is useful for checking purposes and can in fact be used without the above program. To do this, make a list of the machine code to be entered, byte by byte; next write alongside each the equivalent ZX80 character, or blank if no such character is available from the keyboard; finally enter this list in a REM statement and POKE the values where blanks were used. This latter method is only recommended to be carried out for short routines which are required to be attached to a main BASIC program. Other methods are shown in the next chapter.

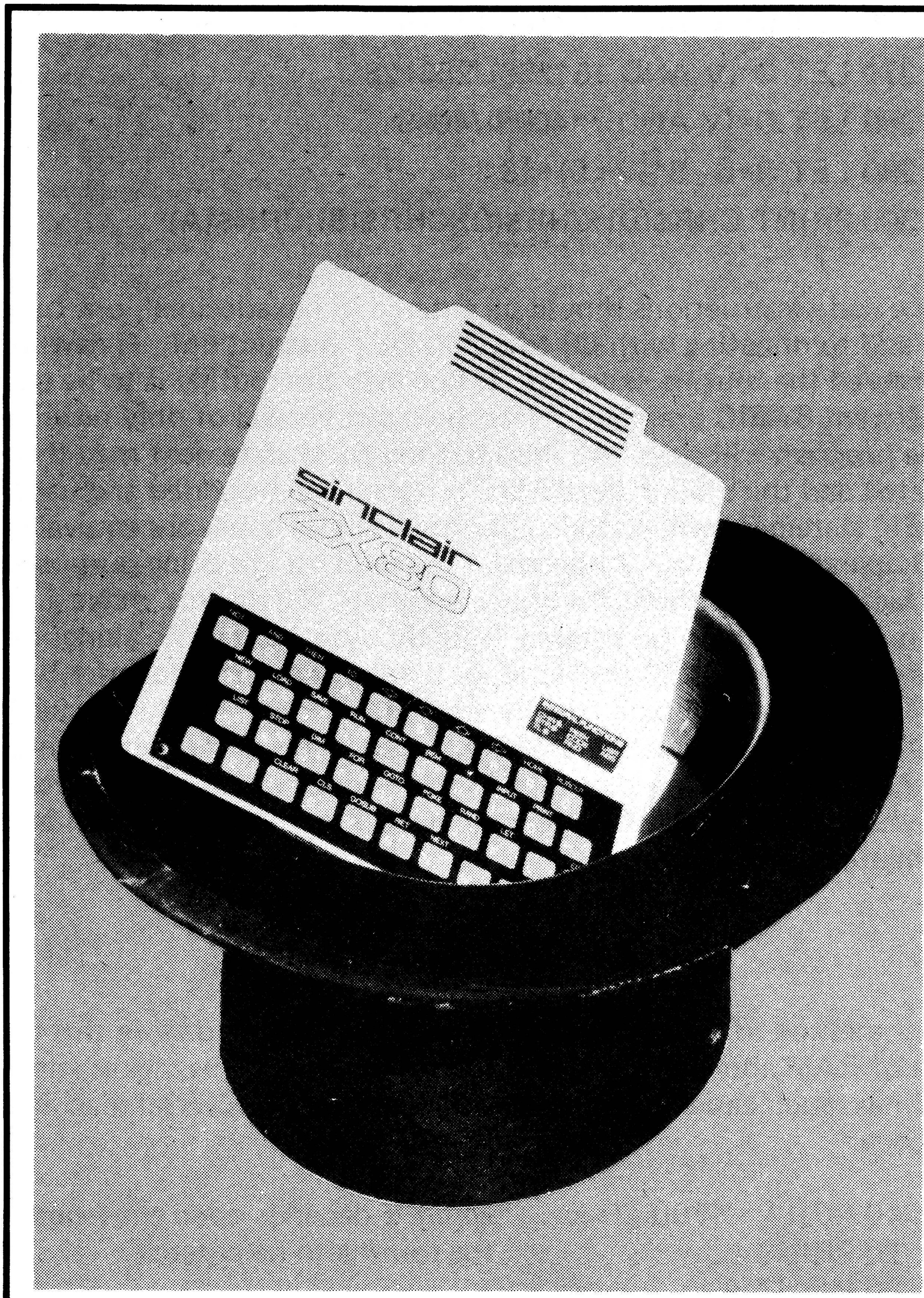
Universal Machine Code Loader

Another method of handling machine code routines is to store them in an area of RAM above the program and display file. This makes entry of routines much easier as shown by the following "universal machine code loader".

```
10 FOR I = 1700 TO x
20 PRINT I,
30 INPUT A
40 POKE I,A
50 PRINT PEEK(I)
60 NEXT I
```

where x depends upon the length of the routine to be entered

A corresponding machine code save routine appears in Section 4.4


sinclair

CHAPTER FOUR – THE ZX80 MONITOR

4.1 INTRODUCTION

The ZX80 is supplied with a 4K preprogrammed Read-Only Memory. This 4K ROM is IC2. The term "4K" means that the chip contains 4096 bytes, each filled with a binary number between 0000 0000 and 1111 1111, or 0 to 255 decimal. This collection of 4096 binary codes forms the program (and associated tables) that runs the ZX80. The program is in Z80 machine code, whilst the tables are lists of numbers that are required by the program.

The monitor can simply be divided into four functional parts:

(i) Locations 0 to 1873 hold an operating system that runs the cassette input and output, the TV screen, and the keyboard.

(ii) Locations 1874 to 1981 hold the main BASIC command table.

(iii) Locations 1982 to 3583 hold the BASIC interpreter.

(iv) Locations 3584 to 4095 hold the character table.

As in most computers, the monitor in the ZX80 can itself be examined using programs that actually are being run under the control of the monitor. In particular, BASIC programs can be written to examine the monitor that will itself be executing these BASIC programs. There are some limitations in this approach but this chapter contains a series of programs that can be used to examine all of the parts of the monitor. A knowledge of Z80 machine language is required for the later programs but there are many interesting features that can be demonstrated using BASIC alone.

4.2 EXAMINING THE MONITOR

The 'PEEK' Instruction

The most useful BASIC instruction for examining the monitor is 'PEEK': by means of PEEK the contents of any specified location in the monitor may be copied and the result used in a BASIC program.

A simple program for beginning to examine the monitor need be no more complicated than the following:—

```
10 INPUT A
20 CLS
30 PRINT
40 PRINT "LOCATION    PEEK VALUE"
50 PRINT
60 PRINT A,,PEEK(A)
70 GO TO 10
```

The 'CHR$' Function

The above program enables the user to access any of the locations in the monitor. However the values obtained do not form a very meaningful list and the program is laborious to use. The next program is a more refined version and gets the computer to produce more information. The CHR$ command is used to produce the appropriate number, letter, graphic or keyword for the appropriate PEEK value.

```
10 GO SUB 30
20 GO TO 100
30 PRINT "  MONITOR LISTING PROGRAM"
40 PRINT " ========================="
50 RETURN
60 FOR I = A TO A+15
70 LET B = PEEK(I)
80 PRINT I,B,CHR$(B)
90 NEXT I
100 PRINT "ENTER THE STARTING LOCATION"
110 PRINT "A NUMBER IN THE RANGE 0-4095"
120 PRINT "ANY LETTER WILL STOP THE PROGRAM"
130 INPUT A
140 CLS
150 IF A<0 OR A>4095 THEN GO TO 10
160 GO SUB 30
170 PRINT "LOC.  PEEK   CHR$"
180 GO TO 60
```

The addition of CHR$ (B) in line 80 forms a method of decoding those parts of the monitor program that use ZX80 character codes. For example, there are tables for all the keyboard characters, the keywords and the integral functions; these and the special characters

inverse K, /, inverse >, inverse S and –

may all be located using CHR$ as above.

Tables and Special Characters

The following notes explain where these are found so that the user may experiment with the above program.

Locations 108–185 This is the keyboard table from which the operating system gets an appropriate value for each key pressed.

Locations 186–315 This is the keyword table which is used to expand the keywords from their single-byte representations (230–254) to a string of characters for printing. The last letter of each keyword has 128 added to its value to mark the end of the keyword. Thus the addition of a line:

```
75 IF B>127 THEN LET B=B–128
```

will make these words more intelligible. Location 315 is of particular interest as it holds the "?" that is printed when the ZX80 expects a keyword and the key pressed does not have a corresponding keyword.

Locations 3008–3052 This is the integral function table that is used for matching the integral functions that are entered letter-by-letter (e.g. PEEK, CHR$); the last letter of each word has 192 added to it to mark its end, and the addition of the lines:

```
72 IF B>191 THEN LET B=B–192
75 IF B>127 THEN LET B=B–128
```

will make these words more intelligible. Between each word are two bytes that form an address for that function.

Location 647 The ubiquitous "inverse K". There is no corres-

ponding "inverse L" as that is obtained by incrementing the K.

Location 1196 — This is the "/" between the run error code and the line number that appears at the bottom left of the screen at the end of a run.

Location 1279 — The current line marker — inverse >.

Location 1312 — The syntax error marker — inverse S.

Location 1706 — The minus sign, "—", that is printed in front of a negative number.

The Character Table

The largest table in the monitor is the character table, which is located from 3584 to 4095. The format of the locations in the table are discussed in Chapters 3 and 5, and a program in section 3.6 shows how words may be printed in large characters using the table. The following program is similar, but enlarges single characters 64 times.

```
10 PRINT "   CHARACTER PROGRAM"
20 PRINT "   ==================="
30 PRINT
40 GO TO 310
50 PRINT "YOU ASKED FOR THE CHARACTER—";
   CHR$(A);"—"
60 PRINT
70 PRINT "LOC.  PEEK    CHARACTER"
80 LET B=3584+A*8
90 PRINT ,,"**********";
100 FOR K=0 TO 7
110 LET D=128
120 PRINT
130 LET C=PEEK(B)
140 PRINT B,C,
150 PRINT "*";
160 FOR I=0 TO 7
170 IF C>D—1 THEN GO TO 200
```

```
180 PRINT " ";
190 GO TO 220
200 PRINT CHR$(128);
210 LET C=C-D
220 LET D=D/2
230 NEXT I
240 LET B=B+1
250 PRINT "*";
260 NEXT K
270 PRINT ,,,"**********"
280 PRINT
290 PRINT
300 PRINT "TO GET THE NEXT CHARACTER"
310 PRINT "PLEASE ENTER A NUMBER'
320 PRINT "BETWEEN 0 AND 63"
330 INPUT A
340 CLS
350 IF A<O OR A>63 THEN GO TO 10
360 PRINT
370 PRINT
380 GO TO 50
```

Making the following changes will show inverse characters:—

```
145 LET C=255-C
320 PRINT "BETWEEN 128 AND 191"
350 IF A<128 OR A>191 THEN GO TO 10
```

The Remaining Tables

In the monitor program there are six other tables. The numbers in these tables are mainly used to form the addresses of sub-routines within the operating system and the BASIC interpreter machine-code program. The first table is 882—897 and it is a table of addresses for the sub-routines associated with the moving of the "cursor".

The details are as follows:—

The Cursor Table

Location	Peek	Location	Peek	Product		Address of Sub-routine	Key
882	169	883	3	3*256 + 169	=	937	↑
884	213	885	2	2*256 + 213	=	725	↓
886	130	887	3	3*256 + 130	=	898	←
888	135	889	3	3*256 + 135	=	903	→
890	185	891	3	3*256 + 185	=	953	Home
892	203	893	3	3*256 + 203	=	971	Edit
894	8	895	4	4*256 + 8	=	1032	N/L
896	149	897	3	3*256 + 149	=	917	Rubout

The BASIC command table is located at 1874—1981, and there is an associated table at 2102—2108.

These two tables give the sub-routine addresses and a command class number (0—6) for all the BASIC commands. However only the following commands yield simple sub-routine addresses.

Command	Sub-routine Address	
LIST	598	
RETURN	2405	
CLS	1863	
SAVE	438	
RANDOMISE	2339	
PRINT	2417	
NEW	0	(What a surprise!)
RUN	2365	
STOP	2350	

Command	Sub-routine Address
LOAD	518
CLEAR	1627
REM	2122

The last three tables are at .—

2723—2732	within BASIC interpreter
3008—3052	Integral function table (Sub-routine addresses after strings)
3359—3384	Relational operators (e.g. +3390,—3385)

but further details would not be helpful at this stage.

The 'USR' Instruction

The BASIC USR command enables machine code progress to be run from BASIC. The programs can either be written and entered by the programmer or selected from the monitor. The last table shows the sub-routine addresses for some of the BASIC commands and the USR command can be used with these addresses, but note that they do not all return successfully to BASIC. For example the SAVE sub-routine can be entered by RUN USR (438) rather than the usual SAVE N/L. RUN USR (456) is dramatic because it is the SAVE routine without the initial waiting loop.

Figure 5 — Monitor Map (as described so far)

Location	Main Blocks	Tables	Sub-Routines
0			Ø NEW
		108 KEYBOARD –185	
		186 KEYWORD –315	438 SAVE
			518 LOAD
512			598 LIST
	Operating System	882 –897 Cursor	
1024			
1536			1627 CLEAR
			1863 CLS
	1873 1874 BASIC Command table 1981		
2048			
			2122 REM
			2339 RANDOMISE
			2350 STOP
			2365 RUN
2560	BASIC Interpreter		2405 RETURN
			2417 PRINT
		3008 Integral –3052 functions	
3072			
		3359-84 Relational operators.	
	3583		
3584	3584 Character table		
	4095		

4.3 USING THE MONITOR SUBROUTINES

So far the monitor has been discussed as a program that can be examined, but of course the greatest benefits come from using parts of the monitor in machine code programs written by the user himself.

As the locations 0 to 1873 form the operating system it is the subroutines in this area that are of the most interest. The BASIC interpreter in locations 1982 to 3583 is a specialised program, and although the arithmetic subroutines can be used, the rest of the program is very complex and will not be discussed further in this volume.

The subroutine memory map (Figure 6) shows the most important subroutines and these are discussed below in detail. The names given to the subroutines have been chosen to show their functions rather than to fit in with any assembly language standard.

4.4 THE PERIPHERAL CONTROLLING ROUTINES

1. The Screen and Keyboard Routine (Locations 316 - 437, CALL at 319)

This very important subroutine checks for keystrokes and, in the absence of a new keystroke, produces a frame for the TV output.

The method of screen maintenance used is unique to the ZX80 and is a complex combination of software and hardware features. The refresh facility of the Z80 microprocessor is used to read the contents of the display file and pass it via the character generator to the TV modulator.

The routine is made up of three functional parts:

(1) The frame counter is incremented

(2) The keyboard is scanned

(3) The current display file is passed to the TV modulator

The routine exits at step 2 if a keystroke has been entered.

The keyboard scan routine is located at 330 to 358 and produces an individual code for every key. The codes are to be found in the BC register pair.

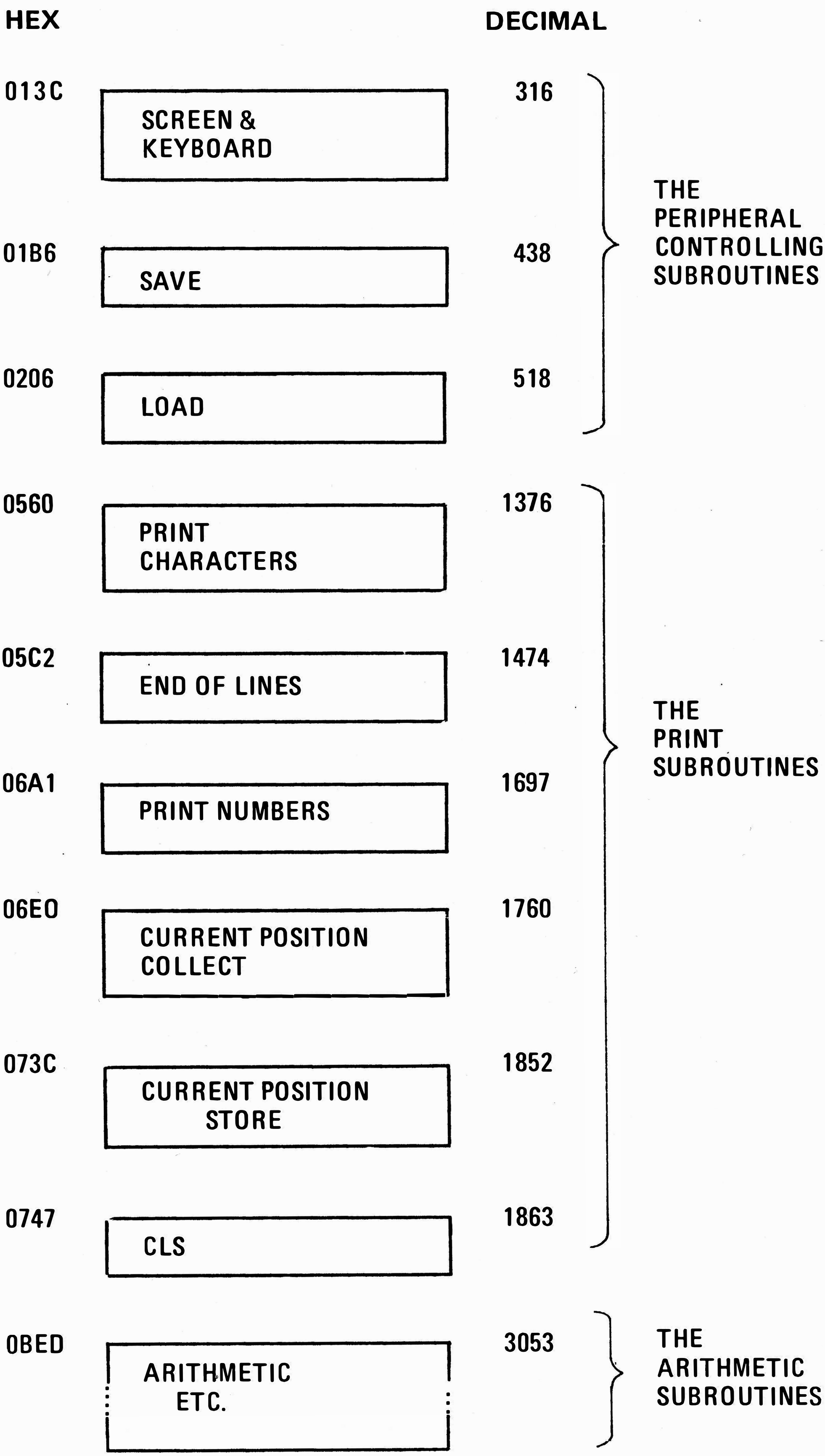


Figure 6 — Subroutine Memory Map

There are several ways of showing these key-codes and the following program gives them as decimal numbers:

```
10 POKE 17000,205
20 POKE 17001,63      Call subroutine screen and keyboard
30 POKE 17002,1
40 POKE 17003,96      LD  H,B
50 POKE 17004,105     LD  L,C
60 POKE 17005,201     Return
```

To obtain a code use the direct command PRINT USR(17000) and press newline: the screen will flash but appear unaltered. The ZX80 is now in the SCREEN & KEYBOARD routine. Next press the required key and the appropriate key-code will appear at the top-left corner of the screen.

To get other codes enter PRINT USR(17000) again. Note that the routine in this form will return to BASIC but if put in a loop it will corrupt its BASIC return. (Section 4.8 solves this problem).

2. The SAVE Routine (Locations 438 - 517, CALL at 438)

This is the subroutine called by the instruction SAVE and is normally used to SAVE BASIC programs on cassette tape. It contains the timing loops for the ZX80 cassette standard, and if a different standard were to be required then it is this routine and the corresponding LOAD routine that would have to be changed.

The subroutine can be interrupted with the BREAK key, as both the 'pause' part and the 'RAM-SAVEing' part contain the following codes that test for the BREAK Key:

```
(From the 'pause' part)
442  LD  A,#7F
444 IN A,FE
446 RRA
447 JRNZ      ..... returns to command mode (i.e. 643) if
                    BREAK detected.
```

The RAM-SAVEing part of the subroutine passes to the cassette output the contents of RAM from its start up to the beginning of the working area pointed to by E-LINE (held in locations 16394 and 16395). It is possible by changing the value in the E-LINE pointer to make the routine pass data to the cassette. However if the E-LINE value is

changed, the SAVE routine must be called from a machine code program, and a corresponding LOAD routine must be used so that the ZX80 is reinitialised successfully.

Most important, in this way it is possible to SAVE machine code programs that have been placed in RAM above the BASIC program and the display file. The following routine is designed to do this, by simply changing the E-LINE value and jumping to the SAVE routine.

```
17000  LD A,0
17002  LD 16394,A
17005  LD A,67
17007  LD 16395,A
17010  JMP 438
```

This changes E-LINE to 17152 (i.e. 67x256 + 0). Values other than 67 and 0 in the first and third lines could clearly be used to produce a different E-LINE value. The above routine can be run by the direct command LET K=USR(17000), assuming it is stored at address 17000.

The 'universal machine code loader' routine (see Section 3.7) can be used to enter this special SAVE routine, using the following decimal values for input:

```
62  0,  50  10  64,  62  67,  50  11  64,  195  182  1
```

3. The LOAD Routine (Locations 518 to 597, CALL at 518)

This is the subroutine called by the instruction LOAD, and is normally used to LOAD BASIC programs from a cassette into RAM. It will only accept tapes using the ZX80 cassette standard.

The LOAD subroutine starts its LOADing into RAM at location 16384 (4000H) and continues to LOAD until the last data has been received from the cassette. In normal working the system variables and the BASIC program and its variables are LOADed and the monitor returns successfully to the command mode entry point at address 643.

However if a special machine code program cassette is LOADed (see previous section) using the LOAD instruction, a successful return to BASIC is no longer assured, and it requires the use of a corresponding special machine code LOAD program to LOAD it and reinitialise the

ZX80 successfully.

The program below moves the LOAD subroutine to near the top of RAM and then alters what was location 516. The effect of this is to force a return to 627, a location in the initialisation block, rather than to 643, the command mode entry point.

```
10 PRINT
20 PRINT
30 PRINT "---MACHINE-CODE-LOADER"
40 PRINT
50 PRINT ,"BY IAN LOGAN"
60 PRINT
70 PRINT ,"----AUGUST 1980"
80 PRINT
90 PRINT
100 FOR I=504 TO 597
110 POKE 16700+I,PEEK(I)
120 NEXT I
130 POKE 17216,115
140 PRINT "1. SET TAPE TO START OF PAUSE"
150 PRINT
160 PRINT "2. PRESS PLAY"
170 PRINT
180 PRINT "3. PRESS [redacted] NEWLINE"
190 INPUT Z$
200 LET K=USR(17219)
```

124 POKE 17285, 52
128 POKE 17285, 67
132 POKE 17285, 52
134 POKE 17286, 67

The above program moves the LOAD routine to 17204 - 17797, and in that particular area the location to be changed is 17216 and the start is at 17219.

4.5 THE PRINT ROUTINES

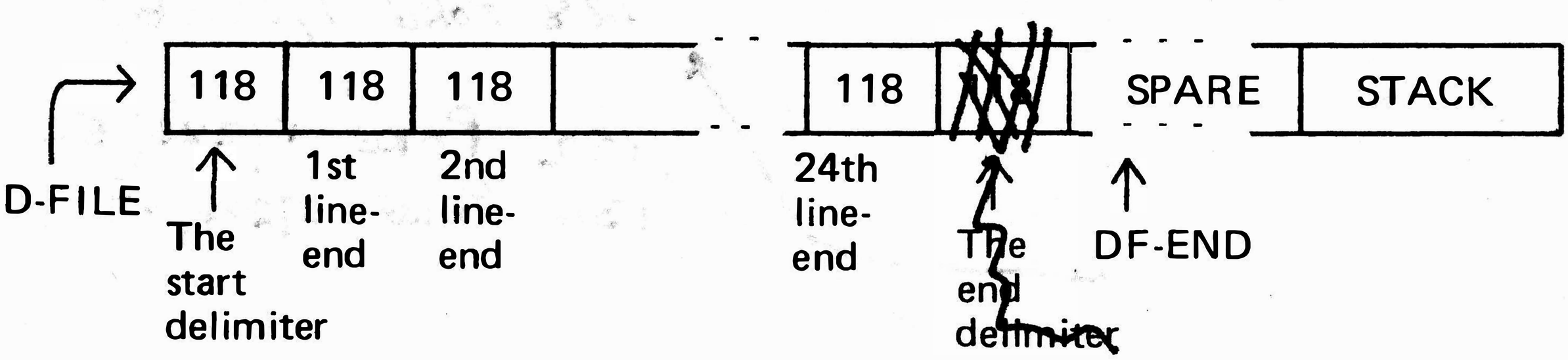
The Display File

In order to use the ZX80 monitor subroutines that control the printing of characters on the TV screen, it is essential to discuss in some detail the concept of the display file as it is used in the ZX80.

The display file is that area of RAM that at a particular moment holds the characters that are to be shown on the next TV frame.

The monitor is so arranged that whenever the 'keyboard and screen' routine is called the display file contains the characters that form 24 lines of display. However the display file has to be constructed character by character and therefore as a program is run (either the machine code monitor program or a BASIC program), the display is built up from its initial zero length to a final length that represents the 24 lines.

The display file lines are filled from the left, with respect to the TV frame, with the ZX80 codes that represent the required characters. An empty line is represented only by a line-end character (118 decimal or 76H), and a completely blank screen will have a display file of 25 line-end characters, i.e. a delimiter at the start and one for each line.



PROBLEM: Write a two line BASIC program that when run produces a completely blank screen. It may help to finish reading the chapter first (answer at the end!).

Constructing the Display File

Every display has to be constructed character by character and the easiest display to discuss is that requested by the programmer as the output from a BASIC program.

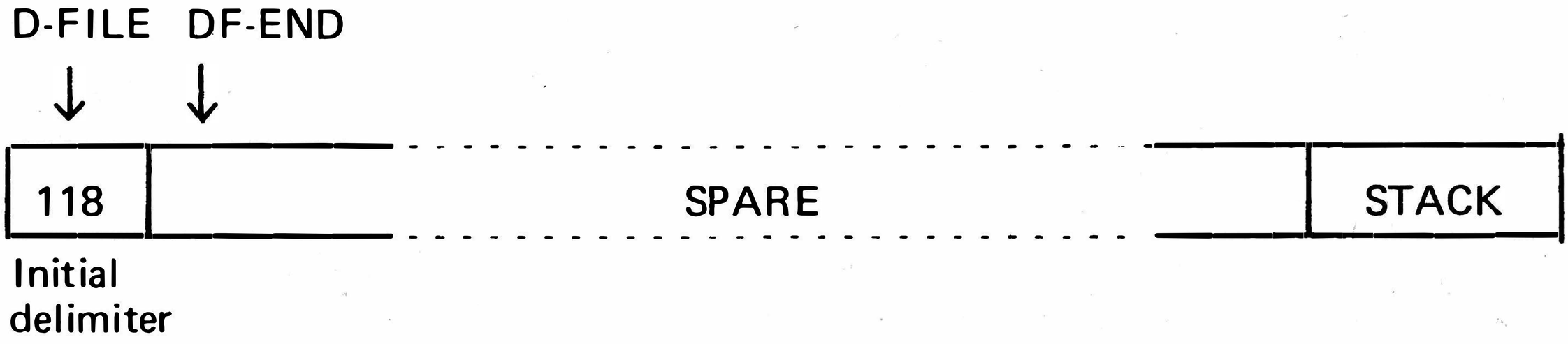
To execute a BASIC program the command RUN is entered and this leads to the following events:

(1) The current value for the start of the display file is collected from the D-FILE pointer.

(2) The initial delimiter, 118, is stored in the D-FILE location.

(3) DF-EA and DF-END are made to point to the next location.

This in effect clears the display file and it now has the format

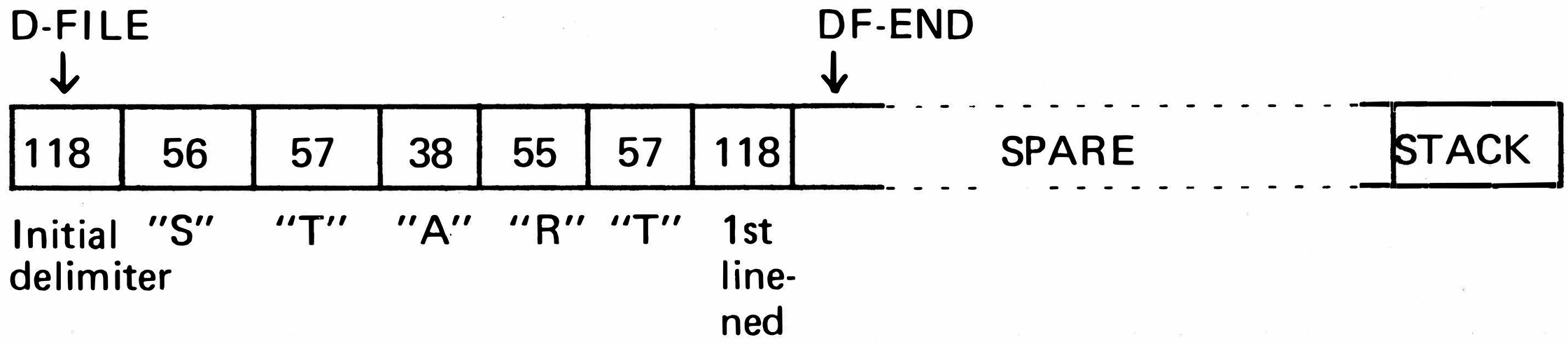


(Note – this is the CLS subroutine as described below)

(4) The monitor now starts executing the BASIC program line by line and whenever a line containing the keyword PRINT is encountered the appropriate characters are added to the display file. For example the line

10 PRINT "START"

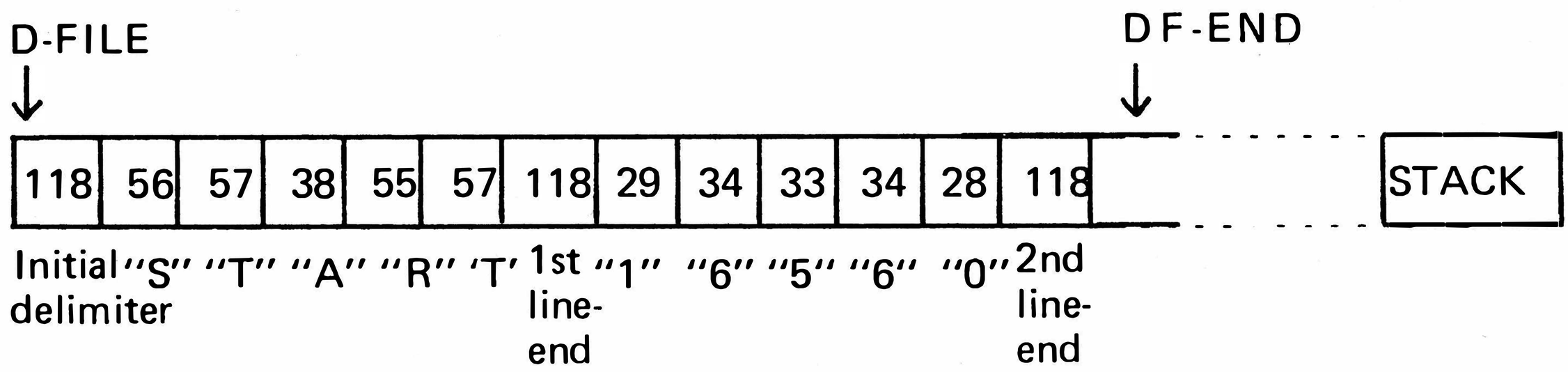
will change the display file to that illustrated below



and the line

30 PRINT A

when A equals 16560, for example, gives



(5) The execution of the program continues until it is stopped by one of the following five conditions:

(a) The last line has been executed
(b) A STOP instruction has been reached
(c) An INPUT instruction has been reached

(d) The display has been filled

(e) There is insufficient room for the display

The effect of all these conditions with respect to the display file then follows the sequence below:—

(1) A number of line-ends are added as required.

(2) The E-LINE is copied into the display file.

(3) More line-ends are added so as to complete 24 lines.

(4) The screen and keyboard subroutine is called, and the completed display file is decoded and shown on the TV screen.

The monitor adjusts the number of lines in the display file so that there is always sufficient room in the finished display for the E-LINE to be added, and so that there is sufficient RAM for the display even if this means showing an error 4 — out of available RAM.

This program shows some of the features discussed above:

```
10 PRINT "START"
20 INPUT A                                    --- Find a start address
30 PRINT A
40 FOR I = A TO A+49
50 PRINT PEEK (I);" ";CHR$(PEEK(I)),  --- Print 50 peek values
60 NEXT I
70 PRINT
80 PRINT "TO CONTINUE PRESS N/L"  - - - Continue loop
90 INPUT A$
100 LET A = A+50                              - - - Increment start
110 CLS
120 GO TO 30                                  - - - Loop back
```

N.B. (1) Input 16561 to show start of display file. Remember that it is the display on the screen that is itself being illustrated by this program.

(2) Input about 16983 to show the line "TO CONTINUE PRESS N/L" as well as the E-LINE and the line-end characters (118) that complete the display file.

The Position of the Display File

An important point to be made is that the display file often moves up

and down RAM during the execution of a BASIC program, and this must be remembered if characters are to be POKEd into a partially completed display file during programs such as games. The problem arises because the variables area is held in RAM below the display file and as the variables area grows or shrinks in size, the display is copied up or down RAM. The following short program has been written to show this at a simple level:

1) Enter:
```
20 GO SUB 100
40 GO SUB 100
99 STOP
100 PRINT "THE DISPLAY FILE IS NOW AT:—"
110 PRINT PEEK (16396) + PEEK (16397)*256
120 RETURN
```

2) RUN: The screen will show that the display file has not moved.

3) Enter:
```
10 LET A=0
30 LET A=30 000
```
or
```
10 INPUT A
30 INPUT A
```

RUN: Again the display file has not moved as all numerical variables are stored in 2 bytes, whether 0 or 30,000.

4) Enter:
```
10 LET A=0
30 LET B=0
```
or
```
10 LET A$ = "AS"
30 LET B$ = "BS"
```

RUN: The display file IS moved as a new variable has been used after the file had already been started.

5) Enter:
```
10 LET A$=" "
30 LET A$= "ABCDE"
```
or
```
10 LET A$ = "ABCDE"
30 LET A$ = " "
```

All these move the display file up or down RAM depending on whether the variable area has been increased or decreased in size.

The Current Display Position

The monitor, whilst constructing the display file, keeps track of the current printing position by using locations 16421 to store the appropriate line number and 16420 to store the appropriate column number.

The following very simple programs show the locations in use.

1) Program to show screen line numbers equalling Y values

```
10 FOR I=1 TO 23
20 PRINT "16421 SAYS THIS IS LINE", PEEK(16421)
30 NEXT I
```

2) Program to show screen column numbers equalling – X values

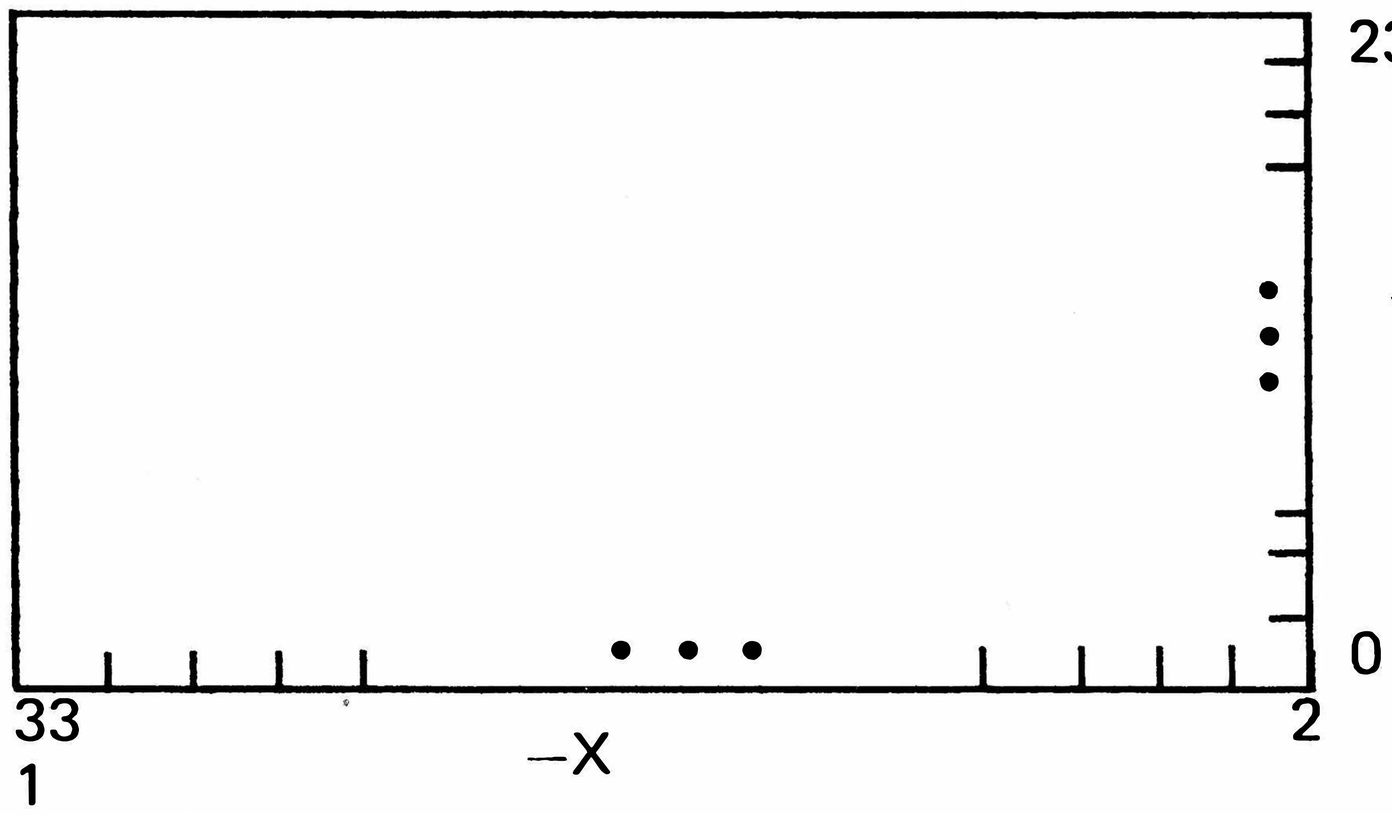
```
10 FOR I=0 TO 21
30 IF I=0 THEN GO TO 70
40 FOR K=1 TO I
50 PRINT " ";
60 NEXT K
70 PRINT PEEK(16420)
80 NEXT I
```

N.B. If line 20 PRINT " "; is added the display is moved over. By increasing the number of spaces in line 20 the column numbers move to the right of the screen. When line 20 contains eleven spaces the program shows a code 1 in the first column. If more than eleven spaces are put in line 20 it will be necessary to decrease the size of the loop, e.g. 10 FOR I=0 TO 10.

The –X, Y Convention

It is useful to use a "–X,Y" convention when discussing screen positions, as the values in 16421 correspond to Y values and the values in 16420 to –X values. This convention divides the screen as shown in Figure 7.

Figure 7 – The –X, Y Screen Convention



The CLS Routine (Locations 1863–1873, CALL at 1863)

This is a very short subroutine and it has been indirectly discussed under the heading 'Constructing the Display File' above. The purpose of the routine is to clear the display file, either after a direct command (no line number) or a CLS instruction. Its function is outlined below –

(1) Loads the HL register pair with the current D-FILE values.

(2) Loads into the D-FILE location, now pointed to by the HL register pair, the initial delimiter 118.

(3) Increments the HL register pair so that it points to the first spare location.

(4) Loads the BC register pair with the values 23 and 33, i.e. Y and –X for the top left-hand corner of the screen.

(5) *Jumps* to the "current position store" routine.

The Current Position Store Routine (Locations 1852–1862, CALL at 1852)

This routine is used to enter line-end characters into the display file. The number of line-end characters added depends on the Y value that is passed to the subroutine in the B prime register.

It is useful at this stage to show the above subroutine being used by discussing the 'end of program' sequence that occupies the locations 1160 to 1209. This sequence is followed when a direct command (apart from RUN and GO TO) has been executed, or the last line of a BASIC program has been executed, or a run error has occurred. The flow chart in Figure 8 shows this part of the monitor.

The Character Printing Routine (Locations 1366–1411, CALL at 1366 to add 28 to character values, or at 1376 to print ZX80 characters

This routine passes characters from the accumulator to the display file. The routine requires that the correct values for the DF-EA, the –X and the Y coordinates are available. It incorporates a test for error 4 (insufficient RAM) and therefore signals an error when the display file reaches the stack. The routine changes the –X and Y values to point to the next screen location, so it is usually followed by a call to the current position store subroutine.

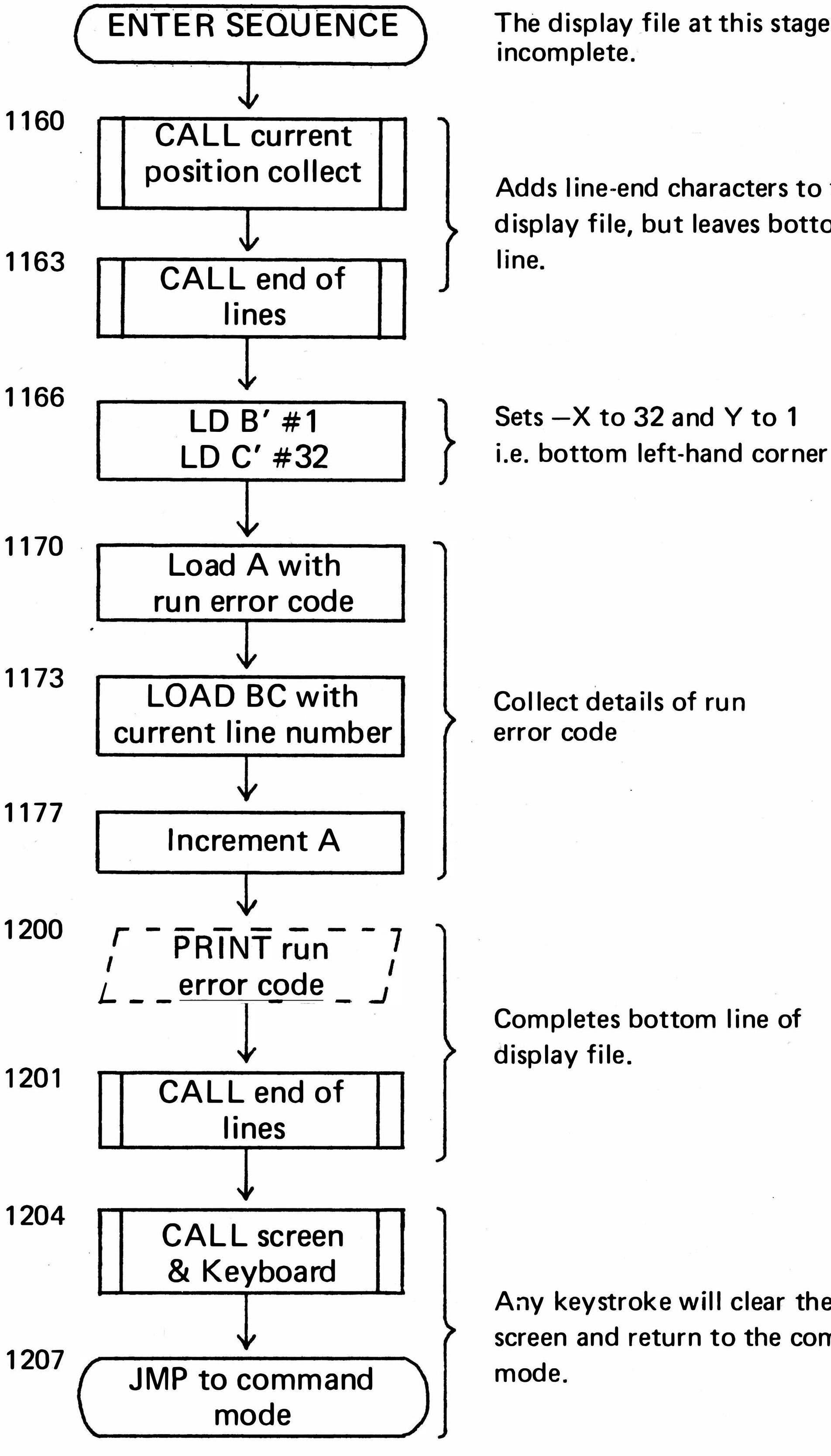


Figure 8 — End of Program Sequence

Character Printing under Machine Code Control

All the necessary subroutines that are required for character printing have now been discussed, but they must of course be used in a strict order. The flowchart below shows a suitable way to organise machine code printing.

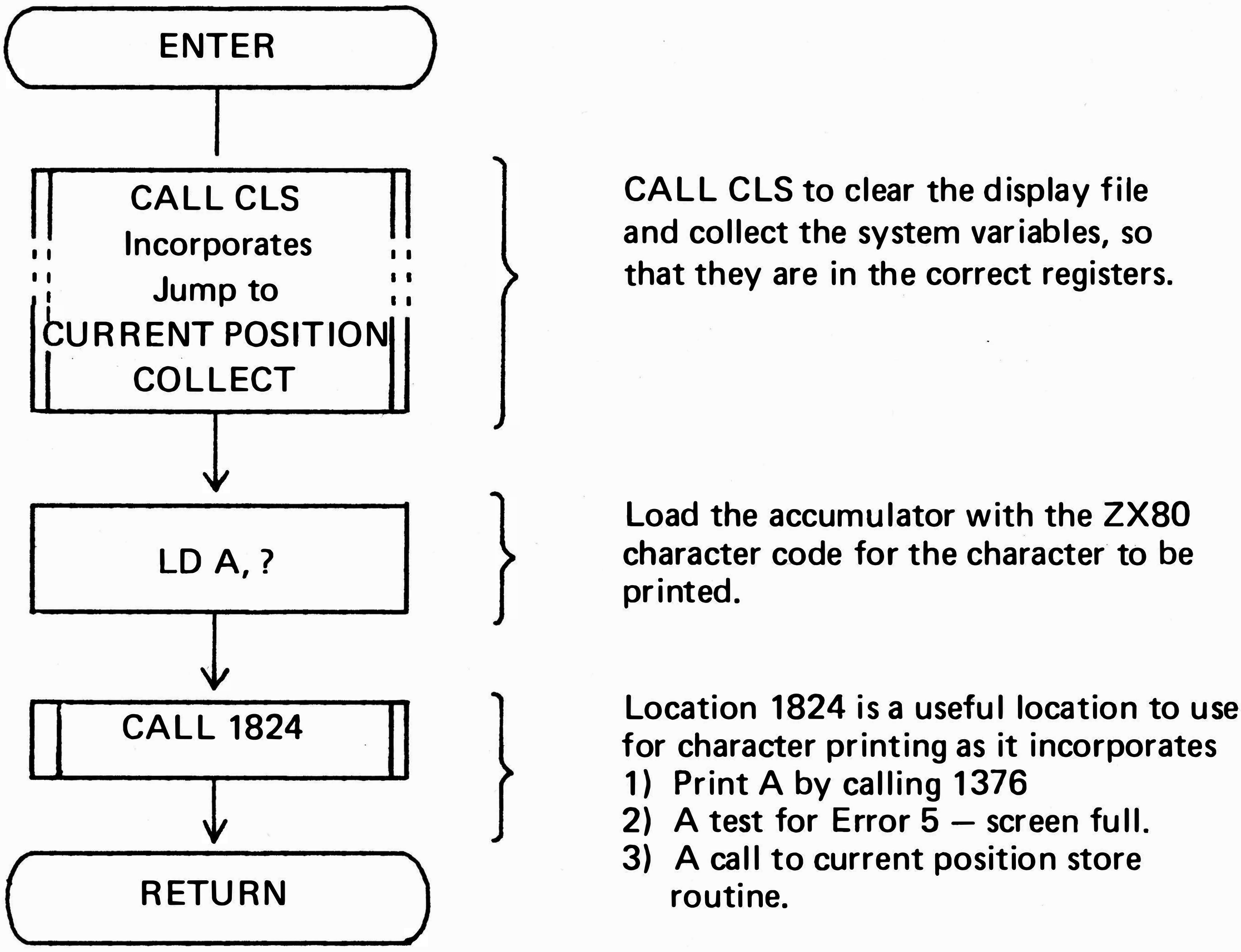


Figure 9 — Machine Code Printing

The following program shows the above system in use. In this example a BASIC loop has been added to print a string of characters, rather than a machine code loop, as this demonstrates the method more clearly.

```
10 POKE 17000, 205
20 POKE 17001, 71
30 POKE 17002, 7
40 POKE 17003, 205
50 POKE 17004, 224
60 POKE 17005, 6
```

10–30: CALL to CLS routine

40–60: CALL to current position collect routine

```
70 POKE 17006, 62      } LD A,1
80 POKE 17007, 1       } load A with character code
90 POKE 17008, 205     } CALL to 1824
100 POKE 17009, 32     } (see flowchart above)
110 POKE 17010, 7      }
120 POKE 17011, 201    } RETURN
```

The above program will run using the direct command LET K=USR17000 but can also be controlled as follows :

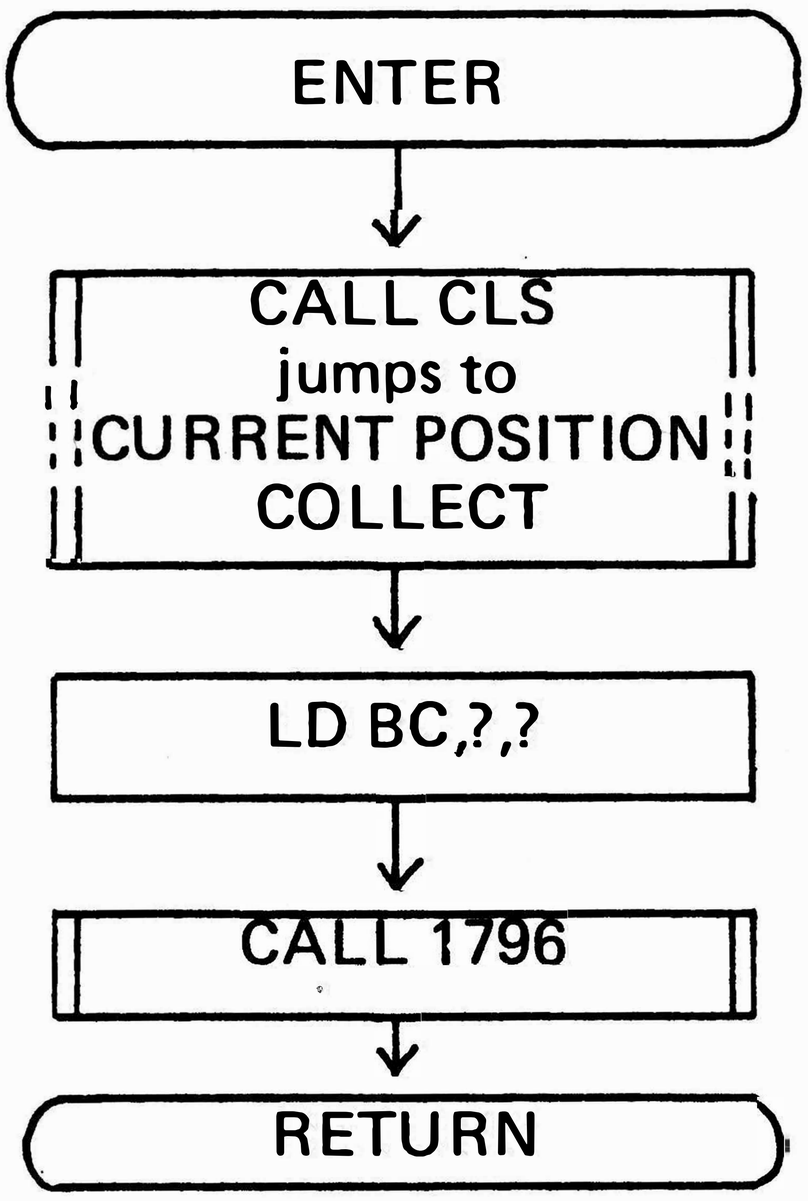
```
200 LET K=USR (17000)      . . . print character 1
210 FOR I = 2 TO 63
220 POKE 17007, I
230 LET K=USR (17003)      . . . do not CALL CLS again!
240 NEXT I
```

The Number Printing Routine (Locations 1697 to 1759, CALL at 1697)

The routine is entered with a number, in two's complement notation, in the BC register pair. It splits the number into its respective digits and calls the character printing routine for each. A minus sign is printed if necessary.

Number Printing under Machine Code Control

The following flowchart shows a system for number printing :



Load BC register pair with the number to be printed, in two's complement notation.

Location 1796 is another useful location as it incorporates
(1) Print numbers (2) A test for error 5
(3) A call to current position store routine

The following BASIC programs shows number printing under machine code control. Once again a BASIC program loop has been added to show one method of printing different numbers

```
10 POKE 17000,205  }
20 POKE 17001, 71  }   CALL CLS
30 POKE 17002, 7   }

40 POKE 17003, 205 }
50 POKE 17004, 224 }   CALL to the current position
60 POKE 17005, 6   }   collect routine

70 POKE 17006, 1   }
80 POKE 17007, 0   }   Load BC with 00
90 POKE 17008, 0   }   i.e. the number 0

100 POKE 17009, 205 }
110 POKE 17010, 4   }   CALL 1796
120 POKE 17011, 7   }

130 POKE 17012, 201 }   RETURN
```

The above will run with the direct command LET K=USR(17000) or it can be repeatedly called from BASIC.

e.g.
```
200 LET K=USR(17000)
210 FOR I=1 TO 20
220 PRINT
230 POKE 17007, I
240 LET K=USR(17003) -- Do not CALL CLS again
250 NEXT I
```

4.6 THE ARITHMETIC SUB ROUTINES

Locations

The arithmetic subroutines in the monitor are as follows :—

Operation	Location
Subtraction	3385
Addition	3390
Multiplication	3396
Division	3472
Exponertiation	~~3340~~ 3440
AND	3509
OR	3576
(RND (X)	3053)

The subroutines must be supplied with the two numbers involved, in the HL register pair and the DE register pair. The answer is returned in the HL register pair. The RND is included here as it uses the multiplication subroutine. It is entered with the X value in the HL register pair and it returns with RND(X) again in the HL register pair. It is noteworthy that LET K=USR(3053) returns random numbers in the range 0—3052 inclusive.

Demonstration of Multiplication Routine

```
 10 POKE 17000, 33      }
 20 POKE 17001, 0       }  LD HL, #00
 30 POKE 17002, 0       }  i.e. Zero
 40 POKE 17003, 17      }
 50 POKE 17004, 0       }  LD DE, #00
 60 POKE 17005, 0       }  i.e. Zero
 70 POKE 17006, 205     }
 80 POKE 17007, 68      }  CALL 3396
 90 POKE 17008, 13      }  multiplication routine
100 POKE 17009, 201     }
```

This can be run by entering PRINT USR(17000) but the following BASIC instructions give an interesting display.

```
200 FOR I=0 TO 22
210 POKE 17001,I
220 POKE 17004,I
230 LET K=USR (17000)
240 PRINT K
250 NEXT I
```

Demonstration of RND Function

A pretty graph of random numbers can be drawn by the following program. In the program USR(3053) is used to generate random numbers in the range 0 to 3052 inclusive. The numbers are then arranged in blocks, i.e. 0—200, 200—400, etc.

```
10 PRINT "RANDOM NUMBER GRAPH"
20 PRINT " ====================="
```

```
 30 PRINT
 40 DIM A(15)
 50 FOR I=1 TO 200
 60 LET R=USR(3053)/200
 70 LET A(R)=A(R)+1
 80 NEXT I
 90 PRINT "3052"
100 LET N=3000
110 FOR I=0 TO 15
120 IF N=0 THEN PRINT "  ";
130 IF N<1000 THEN PRINT ":";
140 PRINT N;"  ";
150 LET N=N-200
160 IF A(15-I)=0 THEN GO TO 200
170 FOR K=1 TO 15-I
180 PRINT "■";
190 NEXT K
200 PRINT
210 NEXT I
220 PRINT "....0....5....1....1....2....2
230 PRINT, ".....0....5....0....5"
```

N.B. Changing line 200 to 200 PRINT A(15–I) will put the actual numbers at the end of the graph lines. Lines 50 to 80 get 200 random numbers and divide them into 16 blocks, A(0) to A(15). Lines 90 to 230 display array A so as to appear as a graph.

This second program takes the 16 blocks as before but this time sorts them into a curve. It can be seen that the RND function always lead to quite an obvious curve!

```
10 PRINT "RANDOM NUMBER GRAPH"
20 PRINT "====================="
30 PRINT
40 DIM A(15)
50 DIM B(15)
60 FOR I=0 TO 200
70 LET R=USR(3053)/200
80 LET A(R)=A(R)+1
```

```
 90 NEXT I
100 FOR I=0 TO 7
110 LET K=I
120 GO SUB 300
130 LET K=15—I
140 GO SUB 300
150 NEXT I
160 FOR I=0 TO 15
170 PRINT "+ ";
180 IF B(I)=0 THEN GO TO 220
190 FOR K=1 TO B(I) —1
200 PRINT " ";
210 NEXT K
220 PRINT "▩"
230 NEXT I
240 FOR I=0 TO 30
250 PRINT "+";
260 NEXT I
270 PRINT "..0....5....1....1....2....2"
280 PRINT ,"...0....5....0....5"
290 STOP
300 LET M=50
310 FOR J=0 TO 15
320 IF NOT M>A(J) THEN GO TO 350
330 LET M=A(J)
340 LET L=J
350 NEXT J
360 LET B(K)=A(L)
370 LET A(L)=40
380 RETURN
```

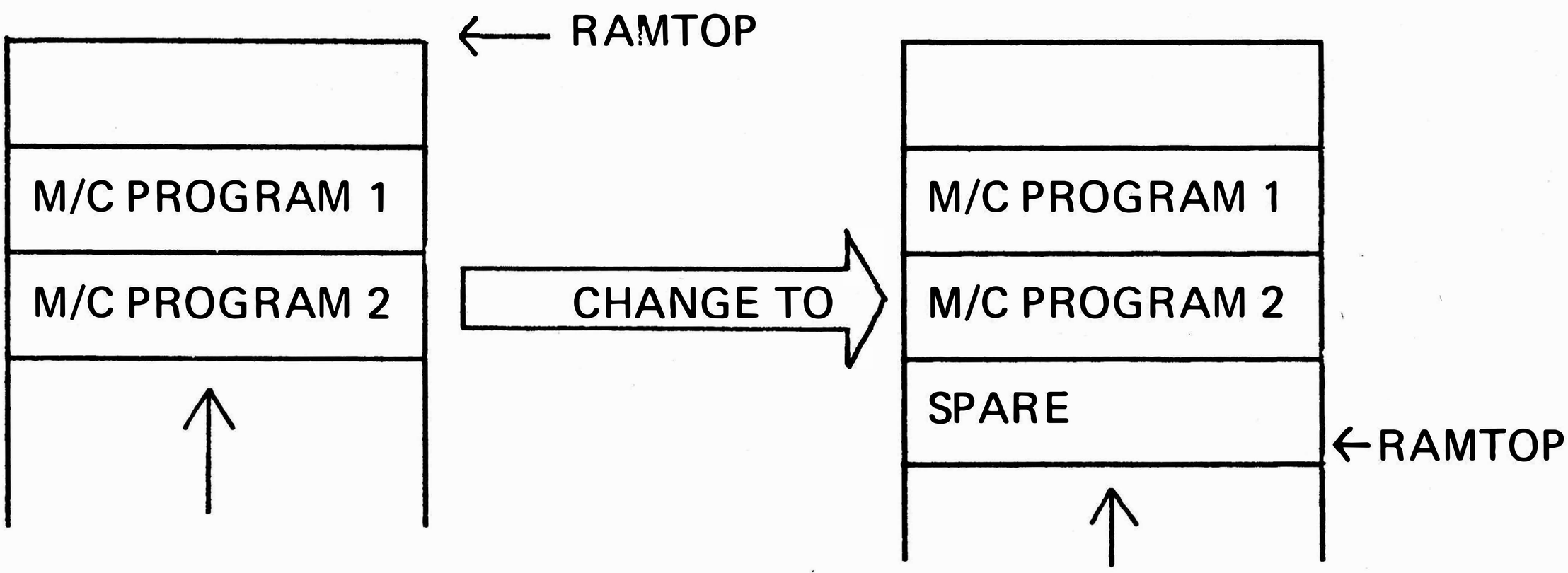
N.B. Lines 60 to 90 — collect random numbers in 16 groups
Lines 100 to 150 — form curve into B(0) to B(15)
Lines 300 to 380 — sort routine for curve
Lines 160 to 290 — GRAPH output

4.7 MOVING 'RAMTOP' DOWN

Protecting Machine Code Programs

In many other computers it is very simple to protect machine code programs by moving the RAMTOP from its usual position above machine code programs to below these programs, so as to make them safe from overwriting. The following section shows how this may be done in the ZX80. A modification to the NEW statement (NEW n) is in fact used to do exactly this in the new ZX80 8K ROM (See Appendix 2).

The desired effect is as follows:

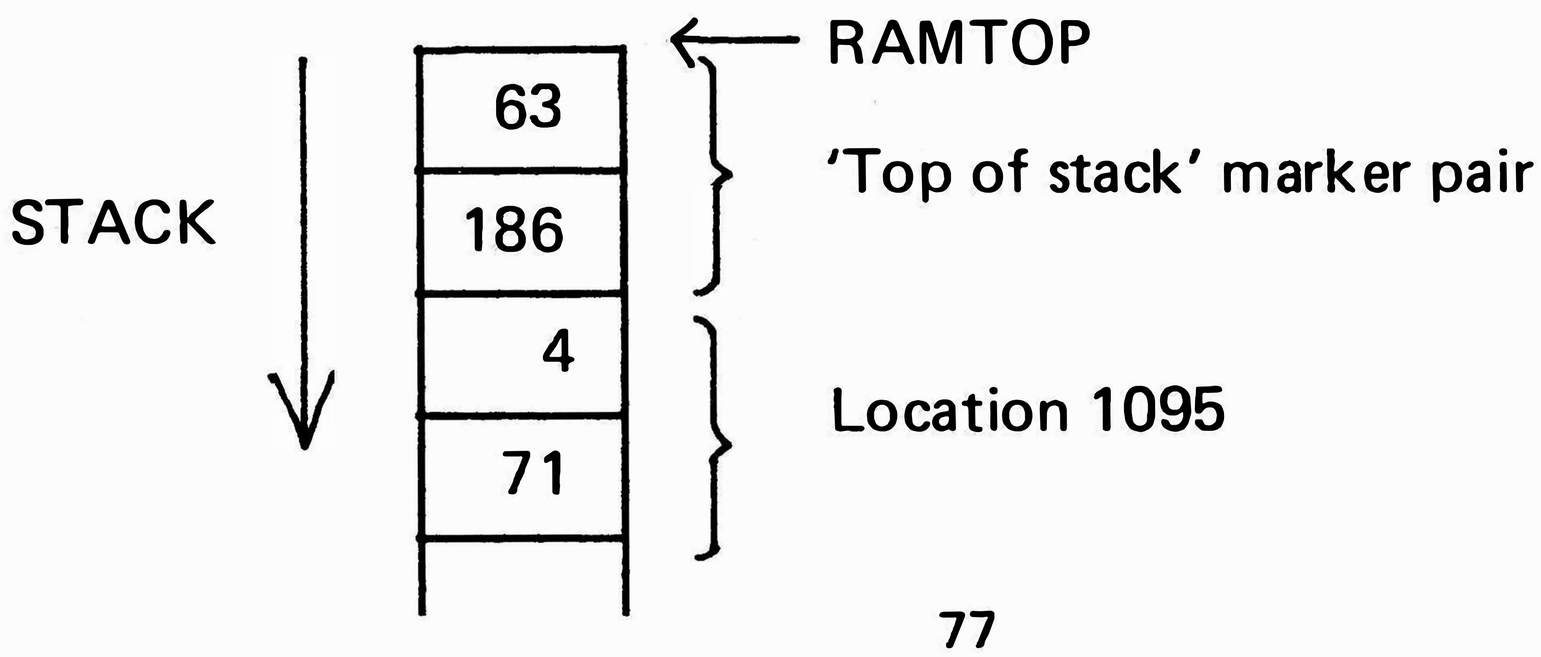


A BASIC program fills RAM from below and can overwrite machine code programs.

A BASIC program reaches the RAMTOP marker before getting to machine code programs.

In the case of the ZX80 the *stack* is located at RAMTOP and below, and it is this "reaching the base of the stack" that stops a program — error code 4. The way to protect machine code programs in the ZX80 system is therefore to move the stack to below the machine code program area.

The usual contents at the top of the stack are obtainable using the PEEK instruction. The top four bytes are:



Program to Show Stack Top

```
10 PRINT "STACK TOP"
20 FOR I=0 TO 7
30 PRINT PEEK(17407-I)
40 NEXT I
```

The value 1095 is the return address for every BASIC line execution, so that if the stack is moved using BASIC, this value must be put on the stack so that the BASIC program will return successfully. The following program shows one method of moving the stack, and involves jumping back to the initialisation routine at 619. It is not possible to transfer stack protection to and from cassette as there is no way of storing the stack pointer as a system variable . (The 8K ROM has not yet been studied sufficiently to ascertain whether it offers this facility). It is however possible to transfer the machine code program that moves the stack.

```
10 PRINT "THIS PROGRAM ENABLES"
20 PRINT "THE STACK TO BE MOVED"
30 PRINT "ENTER NEW STACK LOCATION"
40 PRINT "I.E. 16900"
50 INPUT A
60 LET H=A/256
70 LET L = A-H*256
80 POKE 17000,33
90 POKE 17001, L
100 POKE 17002,H
110 POKE 17003,195
120 POKE 17004,107
130 POKE 17005,2
140 LET K=USR(17000)
```

THIS PROGRAM WHEN RUN, DOES DELETE ITSELF.

4.8 MOVING DISPLAY ROUTINE

Finally, what every ZX80 user has been waiting for – moving displays. The following program allows the user to generate several screen-fulls of output and the ZX80 cycles through them. To run the program enter

1) LIST

2) LIST 400 — to obtain a small display file

3) RUN 200

Enter break to stop.

To obtain further pages of display enter similar sequences to 200–290 and 300–390.

```
 5 LET K=USR(17350)
 6 RETURN
10 FOR I=316 TO 437                  } moves screen & keyboard
15 POKE I +16900,PEEK(I)             } routine into RAM
20 NEXT I
30 POKE 17388,255                    . . . timing counter (number of
                                           frames
32 POKE 17219,24                     } jump out
34 POKE 17220,239                    }
36 POKE 17204,58                     } read counter
38 POKE 17205,186                    }
40 POKE 17206,67                     }
42 POKE 17207,61                     }
44 POKE 17208,200                    }
46 POKE 17209,0                      }
48 POKE 17210,0                      }
50 POKE 17211,50                     }
52 POKE 17212,186                    }
54 POKE 17213,67                     }
56 POKE 17214,24                     } jump back in
58 POKE 17215,7                      }
60 POKE 17350,205                    } the control routine.
62 POKE 17351,67                     } (lines 70–72 specify the
64 POKE 17352,67                     } address of the line-end
66 POKE 17353,33                     } character in line 5)
68 POKE 17354,55                     }
70 POKE 17355,64                     }
72 POKE 17356,34                     }
74 POKE 17357,38                     }
76 POKE 17358,64                     }
78 POKE 17359,201                    }
```

```
 80 RETURN
200 GO SUB 10
210 CLS
220 PRINT "***HELLO***"
230 POKE 16421,PEEK(16421)+1        remove last line
240 FOR I=1 TO 20                   } fill file
250 PRINT                           }
260 NEXT I
290 GO SUB 5
300 GO SUB 10
310 CLS
320 PRINT "***GOODBYE***"
330 POKE 16421,PEEK(16421)+1
340 FOR I=1 TO 20
350 PRINT
360 NEXT I
390 GO SUB 5
400 GO TO 200
```

The above program, just as much of this chapter, has been written for a 1K ZX80, but "moving display" programs really do look better when relocated for a 4K machine.

ANSWER TO PROBLEM ON PAGE 64

The following two line BASIC program produces a completely blank screen.

```
10 PRINT "A"
20 POKE 16446,118
```

Supplementary Problem: Give a proof that the above program does what you think it does. Answers on a postcard please to Ian Logan, c/o LINSAC!

CHAPTER FIVE – CONSTRUCTION & HARDWARE

5.1 INTRODUCTION

Construction

This chapter does not attempt to give full constructional details on the ZX80, to do so would be to duplicate Sinclair's assembly instructions. It is however intended to complement them.

The kit is aimed at the person with a reasonable amount of constructional experience, it is not for the complete novice. The authors recommend that those who do not possess such experience do not try to gain experience by building the ZX80. A ready assembled version is available at the time of printing for £99.95 which may save you the trouble of sending the kit back to Sinclair if it does not work, as well as the cost of repair.

However for those who desire a challenge, buying the kit is a good way of saving yourself £20, which will pay for a power supply and a generous amount of C12 tapes. If, after all your efforts, the kit refuses to work, Sinclair will get it going for you for a minimum charge of £10. Please note however that Sinclair will NOT assemble or complete the assembly of a ZX80 kit.

Apart from your ZX80 all you will need is a power supply, available from Sinclair for £8.95 – or build your own, a television set and a cassette recorder. Join these together, with the leads supplied and you have a complete home computer, waiting to be programmed.

5.2 ASSEMBLY

Before attempting to assemble the kit we recommend reading through Sinclair's assembly instructions at least twice, or until you are familiar with all the steps in the construction of the kit. Sinclair have, we are happy to say, provided instructions that could be called "Noddy builds a computer". They have kept the number of things to do to a minimum, which makes construction of the kit a pleasure rather than an endurance course. We took about three hours to build the kit but would suggest

that five hours will be about average for its construction. This is going to be one of its greatest selling points as you can be programming in BASIC within five hours of the kit's arrival.

Provided you follow Sinclair's instructions very little can go wrong with the construction of the kit. They really have tried to make construction easy and to enable you to get the right bits in the right places. The assembly instructions include a components list, a component location diagram, a circuit diagram and fairly explicit constructional details.

The kit itself is of a very high standard, the keyboard being a part of the main board. The PCB is double sided with plated through holes. If you can remember as far back as Sinclair's "Cambridge" calculator kit, as we can, just looking at the board makes you realise how quickly things have developed in the last six years. We recommend inspecting the board visually for solder "bridges" or breaks in any of the tracks. We also suggest checking each of the components as much as you can before insertion. Resistors can be wrongly coded and diodes can be short circuit or open circuit, etc. Although we did not find faults on the board or any components, checking of both may save you time and possibly money later. Once you are satisfied that all is well, construction may commence.

When everything is soldered onto the board we suggest testing the board prior to inserting the socketed i.c.'s. Sinclair have kept the amount of tools necessary for the construction of the kit to a minimum and it can be built with those tools, but a greater degree of confidence can be added if you have or have access to a multimeter and possibly an oscilloscope.

Once you have finished soldering, check the board again visually for any solder "bridges" or holes left unsoldered which have component legs in them. Next power up the board and check the voltage going into the voltage regulator (i.c. 22) and the voltage coming out. The input voltage must be at **least** seven volts, if it is not, the voltage regulator will not work properly. The output voltage must be between 4.75 and 5.25 volts, any voltage outside of these two extremes is not acceptable as it will damage the i.c.'s. If the output voltage is found to be less than 4.75 volts then check the board again for short circuits between the 0 volt and the +5 volt supply rails. If the output voltage is inside these two limits then check that power actually gets to the correct pins on the top of each i.c. socket. (The circuit diagram shows where the +5 volt and 0 volt pins are on each i.c.). When you have established that power

actually goes to each i.c. socket then after powering down the board the i.c.'s can be inserted. Make **sure** that all are in the correct way round.

It is now time to make connection to the T.V. — turn on the T.V. and let it warm up, then connect the video lead from the UHF modulator into the T.V. aerial socket. Turn on the ZX80 and the cursor (a black square with a white K in it) should appear on the bottom left hand corner of the screen. Sinclair specify that the T.V. should be tuned to UHF channel 36 but it may vary slightly and they suggest that you try anywhere between channels 33 and 39 until you get the best picture.
N.B. brightness and contrast controls may need to be varied to give the best character definition.

Hopefully your kit will work first time (ours did) if it does not **do not panic.** Read through the section on faults in the assembly instructions and see if any of the faults listed there corresponds with the fault you have. If none of them do and the power supply voltages are correct then without at least an oscilloscope the things you will be able to do will be very limited. If you do have access to an oscilloscope you can check that the Z80 (i.c. 1) has the correct clock signal going to it. The clock signal Ø may be found on pins 10 and 12 of i.c. 15. Unless you have a wide-band oscilloscope this will look like a 3.25 MHz sinewave, it should in fact be approximately a square-wave varying between normal TTL levels. This signal should also be found on the Z80 (i.c. 1) pin 6. If the clock signal is found on pin 6 but operation is still not evident then the levels on four inputs to the Z80 should be checked. The voltage levels on $\overline{\text{RESET}}$, $\overline{\text{BUSRQ}}$, $\overline{\text{WAIT}}$ and $\overline{\text{NMI}}$, pins 26, 25, 24 and 17 of i.c. 1 should all be greater than 2.4 volts. If any are below 2.4 volts then checks should be made on the pull-up resistors, R21, R22, R23 and R12. All of these signals are brought out to the back edge connector of the ZX80 so if a fault exists check also that there are no solder "bridges" on the edge connector.

If after trying the above the fault still exists, and you have checked that all the i.c.'s are correctly seated in their correct sockets, then we recommend sending your kit back to Sinclair as the fault, most probably, is beyond repair without more specialised equipment.

5.3 HARDWARE

The ZX80 is controlled by an NEC 780-1 microprocessor which is a copy of the very successful Z80, hence the name. The CPU is used to control

all the functions of the ZX80, including driving the T.V. display and the cassette recorder. Sinclair have opted for this approach in order to keep the system compact, by doing so they have saved i.c.s, board area and also costs which is reflected in the price of the machine. The CPU runs at 3.25 MHz which as it happens is a convenient multiple of the T.V. line frequency (15.625 KHz). Since the CPU drives the display then whilst the CPU is processing data the display disappears, i.e. when data is input from the keyboard, calculations are taking place or data is being loaded from or stored on a cassette. We found the "blinking" of the display when accepting input from the keyboard a positive help as it signifies acceptance of each keystroke.

Display

The display itself is very clear, the picture is rock steady and there is the option of having white characters on a black background. For those who want this option to be switchable a one pole two way switch inserted via a couple of wires in place of links A—C and A—B on the underside of the main PCB will give you this option at your fingertips. It is advisable however to make the wires as short as possible.

A one volt composite video signal is available for use with a monitor if it is required. This signal normally feeds the UHF modulator but could be taken out and fed, via a buffer circuit, to a monitor. The authors feel this is probably unnecessary as the standard of the display is already very high. Details of this modification are to be provided by Sinclair at a later date.

The display consists of 24 lines each of 32 characters, 768 displayable characters thus being possible. It is not memory mapped, in fact it is treated like a serial file. It is possible, using POKE, to put characters into the display file area, however **all** variables must be assigned values **before** you do this. This however does not make moving graphics a possibility since while the CPU is processing data the display is blanked.

Character Set

The number of displayable characters can be extended by using the character generator contained in the main ROM in your own program. The character generator is located at addresses OEOO to OFFF hexa-decimal. Each character takes up eight bytes, as shown in the following example.

Address	Data
OEO8	00
OE09	14
OEOA	14
OEOB	00
OEOC	00
OEOD	00
OEOE	00
OEOF	00

These eight bytes correspond to character number 1 in the character generator set which is ''

i.e.

OEO8	0	0	0	0	0	0	0	0	
OE09	0	0	0	1	0	1	0	0	
OEOA	0	0	0	1	0	1	0	0	
OEOB	0	0	0	0	0	0	0	0	
OEOC	0	0	0	0	0	0	0	0	which = ''
OEOD	0	0	0	0	0	0	0	0	
OEOE	0	0	0	0	0	0	0	0	
OEOF	0	0	0	0	0	0	0	0	

Bits set to logic 1 are normally displayed as black (white when using inverse video). The graphics characters use the full eight by eight matrix, whereas the alpha-numeric characters only use a matrix of six by eight dots. Larger characters can be obtained in your own programs by looking at each byte of the character in turn and displaying any one character of the normal character set when a bit is set to logic 1 in the byte, otherwise display a space. (See Section 3.6)

Use of the memory display program in Chapter 6 is very useful when looking at the character generator section of the ROM as the display format is as shown below :—

Address	Data
OE58	00 1E 21 78 20 20 7F 00
OE60	.....
OE68	... etc.

The format used is the address (in hex) followed by eight data bytes (in hex) which in this case correspond to a complete character, actually a £ sign. Use of this format allows you to see one "character" of the character generator on each line of the display.

Keyboard

The keyboard of the ZX80 is attached to the main PCB. It is waterproof and chemical proof so if you spill your coffee on the ZX80 you will not have the problem of the keys being stuck. It is touch sensitive and has a standard typewriter layout, though compressed. The key symbols are printed on the underside of the plastic coating so they will not rub off with wear.

It is recommended that when using the keyboard a rolling or wiping action is used instead of the more usual tapping movement as this tends to produce, much more consistently, correct entry of characters.

The ZX80 switches the function of the keys on the keyboard between keywords and characters as it checks the syntax of your statements. Thus at the command level typing R will produce the command RUN whereas if you were typing in a program line and you used the key R **after** a command the display would show R as in :—

```
100 LET R = A*B
```

Cassette Interface

The cassette interface data format is unique to the ZX80 and is not compatible with any other computer system. The ZX80 does not use the more popular Kansas City cassette interface as to have done so would have required more integrated circuits and more board area. However it does work quite well. We initially had some problems with our machine not loading programs when the voltage regulator became hot. This has been alleviated by using a thicker insulator between the heatsink and the board, thus cooling the board and the surrounding integrated circuits. If you do experience this problem turn off the machine and leave it to cool down for five minutes or so, after which time it will work quite happily.

Connection is made to a standard cassette recorder via two 3.5mm jack

sockets. The leads provided with the ZX80 have matching 3.5mm jack plugs but if your cassette recorder has a DIN connection then a DIN jack connector can be bought from most electrical stores. However, since the DIN connection gives a lower output signal and the ZX80 is somewhat ''hard of hearing'', such connections are not recommended.

Two other points have been noticed about the cassette interface, they are:—

1. Some recorders may record noises when the CPU is supposed to be sending out five seconds of silence. This may be due to the ''ear'' jack being in circuit. It is therefore recommended that the ''ear'' jack is disconnected before issuing the SAVE command.

2. LOADing of files from a cassette works better if the tape is moved into the silent part of the file, preceding the file header record, before issuing the LOAD command.

At present there are no file handling facilities available other than LOAD and SAVE. However a new ROM is being developed that will include such routines and will enable you to name your files and write them to cassette and possibly discs.

Mains Adaptor

The mains adaptor available from Sinclair gives an output of about 600mA at 9 volts DC. This voltage is unregulated as the ZX80 has its own on board voltage regulator (ic 22). Sinclair's adaptor is handy, but if you want to make your own, any power supply that gives an output of about 9 volts at 600mA will suffice. N.B. If you do buy the Sinclair adaptor do not leave it on **all** day: we did and blew the thermal fuse that is inside the adaptor.

Edge Connector

At the rear of the ZX80 there is a forty six way double sided edge connector, (standard 0.1'' pitch). The signals that are brought to this connector are;

8 data lines, $\overline{D0}$ to $\overline{D7}$

16 address lines, A0 to A15

13 control lines, $\overline{INT}$, $\overline{NMI}$, $\overline{HALT}$, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$, $\overline{BUSAK}$, $\overline{WAIT}$, $\overline{BUSRQ}$, $\overline{RESET}$, $\overline{MI}$, $\overline{REFSH}$

1 3.25 MHz clock

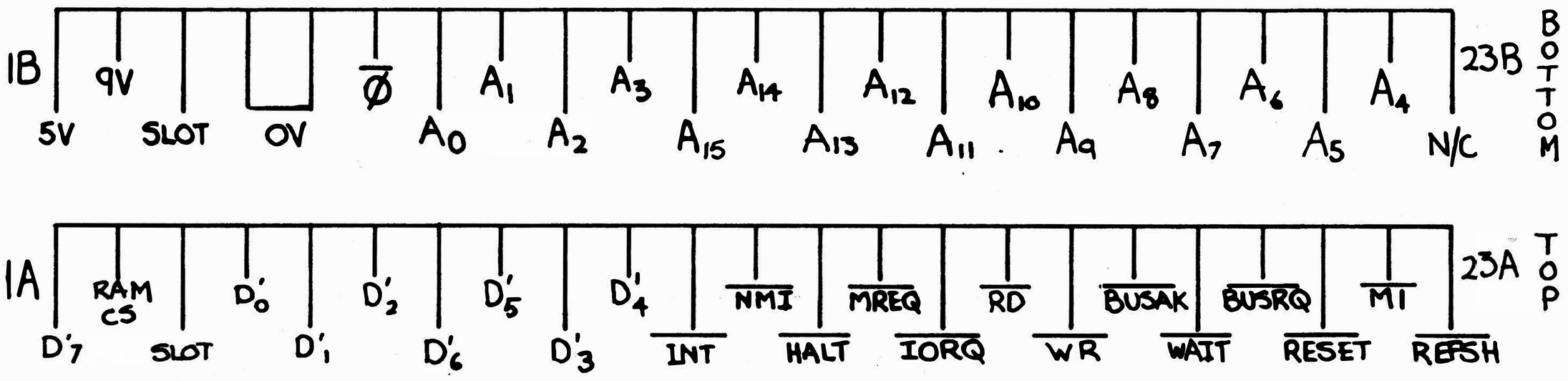
1 chip select for external RAM, RAM $\overline{CS}$

1 5 volt stabilised power supply line

1 9—11 volt unstabilised power supply line

2 0 volt lines

The locations of which are shown in the following diagram.



The edge connector is used for the memory expansion boards. It should be noted that the data and address lines are not buffered and as yet there are no I/O ports on the ZX80. If it is desired a RESET switch can be attached to the edge connector between $\overline{RESET}$ and 0 volts. It should be a momentary action push to make switch. This switch has the same effect as pulling the power supply plug out of the ZX80. It should only be used when control of the machine has been lost as both program and variables are destroyed.

An explanation of all of the control lines has not been given as it would be duplicating data which is much more comprehensively dealt with in other more specialised publications. A list of recommended books can be found at the end of this chapter.

ROM

At present the operating system, monitor, BASIC interpreter and character generator are all contained in a 4K byte ROM, (See Chapter 4). This ROM is pin-compatible with the Texas Instruments TMS2532 EPROM so if you have EPROM programming facilities it would be possible to write your own programs and pop it in place of the system ROM. Shortly Sinclair will be bringing out an 8K byte ROM that will

overcome the shortcomings of the system at present. It will contain the trigonometrical, logarithmic and floating point arithmetic functions that are missing at present and will also contain file handling routines. Thus the ZX80 will be able to read and write files to tape, calling them by name. It will also, it is hoped be able to access discs. The new ROM is a plug in replacement for the existing ROM and will cost around £20.

RAM

The ZX80 is supplied with 1K byte of RAM as standard. The i.c.s used are standard 1024 x 4 bit NMOS Random Access Memories. Extra RAM can be added to the system via the rear edge connector. Science of Cambridge are selling memory expansion boards which each take a further 3K bytes of RAM, the memory chips being sold separately in 1K byte increments. Several other companies are now offering memory expansion for the ZX80, both 4K byte and 16K byte expansions being offered at the time of writing.

I/O

At present Sinclair are not marketing an I/O bus for the ZX80. It is however possible to attach one via the edge connector. I/O will most likely be memory mapped, i.e. input or output ports will look like memory locations to the microprocessor. As the ZX80 uses $\overline{IORQ}$ as part of the decoding for the keyboard and some address lines are used for different functions you must be very careful if you try to attach a normal I/O bus to the ZX80. It would seem that by far the easiest way of attaching I/O is to make it memory mapped.

ZX80 IC LIST

IC	Type No.	Functional Description
1	780-1	Z80 equivalent microprocessor
2	2332	4K byte ROM
3	2114	1K x 4 bit RAM
4	2114-1	1K x 4 bit RAM
5	74LS373	Octal transparent latch
6	74LS157	Quad 2 to 1 line data selector
7	74LS157	Quad 2 to 1 line data selector
8	74LS157	Quad 2 to 1 line data selector
9	74LS165	8 bit shift register (P.I.S.O.)
10	74LS365	Hex bus driver 3-state
11	74LS00	Quad 2 input NAND gates
12	74LS00	Quad 2 input NAND gates
13	74LS04	Hex inverter
14	74LS05	Hex inverter (open collector)
15	74LS05	Hex inverter (open collector)
16	74LS10	Triple 3 input NAND gates
17	74LS32	Quad 2 input OR gates
18	74LS32	Quad 2 input OR gates
19	74LS74	Dual D-type flip flop
20	74LS86	Quad exclusive OR gates
21	74LS93	4 bit binary counter
22	7805	+5 volt voltage regulator

CHAPTER SIX — ZX80 PROGRAMS

The following pages contain a selection of programs from the current LINSAC program library.

All programs are available on C12 cassettes and copies of the program catalogue are available from LINSAC.

REF. NO. E002 DOCUMENTATION

PATTERN RECOGNITION

PROGRAM DESCRIPTION

Tests the player's ability to recognise the number of a random collection of dots, within a time limit. There are five questions. The player specifies the maximum number of dots required. At the end the player is notified of his score.

Glenn Doman uses similar methods to teach children mathematics — see his book "Teach Your Baby Maths".

INSTRUCTIONS

Type RUN and NEWLINE once program is loaded.

The program first requires that the maximum number of dots wanted should be entered. Any value outside the range 10 to 100 is ignored.

The player requests each question when prompted by pressing the NEWLINE key (N/L).

Once the pattern has been displayed the player has ten seconds to estimate the number of dots. After entry of his guess he is notified of its accuracy or otherwise, unless he is outside the time limit. At the end of five questions the player may obtain his overall score by pressing NEWLINE when prompted.

LIST OF VARIABLES

X	=	the number of correct answers
Y	=	the number of answers within 5 of the correct numbers
Z	=	the number of answers within 10 of the correct numbers
M	=	the maximum number of dots, chosen by player (10 – 100)
G	=	sequence number of the go
N$	=	dummy variable
R	=	the actual number of dots
I, J	=	loop counters
N	=	the number estimated by the player
D	=	the modulus of the difference between R and N

SAMPLE RUN

```
o DOTS o
MAX (10 – 100)  =  99
      5 QUESTIONS
N/L FOR GO 1

Newline entered, screen clears
A collection of dots is displayed
HOW MANY?
68 entered, screen clears
68
YOU WERE 3 OUT
THERE WERE 65 DOTS

N/L FOR GO 2
Newline entered, screen clears
A collection of dots is displayed
HOW MANY?
55 entered, screen clears
55
YOU WERE 30 OUT
THERE WERE 85 DOTS

N/L FOR GO 3
Newline entered, screen clears
A collection of dots is displayed
```

SAMPLE RUN continued

```
HOW MANY?
45 entered, screen clears
45
YOU WERE 4 OUT
THERE WERE 41 DOTS

N/L FOR GO 4
Newline entered, screen clears
A collection of dots is displayed
HOW MANY?
42 entered after some delay, screen clears
TOO SLOW
THERE WERE 44 DOTS

N/L FOR GO 5
Newline entered, screen clears
A collection of dots is displayed
HOW MANY?
13 entered, screen clears
13
o  CORRECT  o
END
N/L FOR SCORE
Newline entered, screen clears
YOU GOT   1 CORRECT
          2 WITHIN 5
          0 WITHIN 10

Program ends
```

PROGRAM LISTING

REF E002

```
2 LET X=0
4 LET Y=0
6 LET Z=0
10 RANDOMISE
20 PRINT ,"DOTS"
25 PRINT "MAX(10-100)=";
30 INPUT M
32 IF M<10 OR M>100 THEN GO TO 30
35 PRINT M
```

PROGRAM LISTING Contined

```
 40 PRINT
 60 FOR G=1 TO 5
 70 PRINT "N/L FOR GO";G
 80 INPUT N$
 90 CLS
100 LET R=RND(M)
110 FOR I=1 TO R
120 PRINT "■";
122 FOR J=1 TO RND(2)
124 PRINT " ";
126 NEXT J
130 NEXT I
150 PRINT
160 PRINT "HOW MANY?"
162 POKE 16414,0
164 POKE 16415,0
170 INPUT N
171 CLS
172 IF (PEEK(16415)*256+PEEK(16414)-4)>500 THEN
    GO TO 225
180 PRINT N
190 IF N=R THEN GO TO 230
192 LET D=ABS(N-R)
195 PRINT "YOU WERE ";D;" OUT"
200 IF D<6 THEN GO TO 250
210 IF D<11 THEN GO TO 280
220 GO TO 300
225 PRINT "TOO SLOW"
227 GO TO 300
230 PRINT "  CORRECT  "
240 LET X=X+1
245 GO TO 310
250 LET Y=Y+1
260 GO TO 300
280 LET Z=Z+1
300 PRINT "THERE WERE ";R;" DOTS"
310 NEXT G
320 PRINT

330 PRINT "END"
340 PRINT "N/L FOR SCORE"
```

PROGRAM LISTING continued

```
350 INPUT N$
360 CLS
370 PRINT "YOU GOT",X;" CORRECT"
380 PRINT ,Y;" WITHIN 5"
390 PRINT ,Z;" WITHIN 10"
400 STOP
```


MEMORY DISPLAY

PROGRAM DESCRIPTION

Displays 80 byte blocks of memory in hexadecimal or character form starting at an address as input.

Contains decimal to hexadecimal conversion routine.

INSTRUCTIONS

Press RUN.

Input starting address between 1 and 32767.

On prompt type NEWLINE for display of next 80 byte block or Z to stop.

N.B. Some time is taken to set up and output each block.

LIST OF VARIABLES

A1	=	nearest multiple of 16 at which memory display to commence.
A	=	pointer to current address being considered
A$	=	dummy variable
X	=	parameter passed to subroutine 300 containing decimal number for conversion
C	=	decimal contents of address A

MEMORY DISPLAY

SAMPLE RUN

```
MEMORY DISPLAY
START ADDRESS (DECIMAL)?
o  HEX OR CHARACTERS? HEX

Screen clears, delay
4000    FF   E1   46   00   06   42   00   00
4008    00   42   25   42   25   42   62   42
N/L (or Z)
Newline entered, screen clears, delay
4010    31   42   15   00   00   00   00   19
4018    00                                  6A
4020    ED
4028    00   etc.
4030    32
• • •
4058    26   31   D9   0F   01   76   00   06
N/L (OR Z)
Z
Stops
```

PROGRAM LISTING

```
2 PRINT, "MEMORY DISPLAY"
3 PRINT
4 PRINT "START ADDRESS (DECIMAL)?";
5 INPUT A1
6 PRINT A1, "HEX OR CHARACTERS?"
7 INPUT A$
8 LET H=(CODE(A$)=45)
9 CLS
10 LET A1=80*(A1/80)
20 FOR A=A1 TO 32767
21 IF NOT A=80*(A/80) THEN GO TO 30
22 PRINT
23 PRINT "N/L (OR Z)"
24 INPUT A$
25 IF A$ = "Z" THEN STOP
26 CLS
```

PROGRAM LISTING continued

```
 28 PRINT "ADDRESS",, "CONTENTS"
 30 IF NOT A=8*(A/8) THEN GO TO 70
 35 PRINT
 40 LET X=A
 50 GO SUB 300
 60 PRINT ,
 70 LET C=PEEK(A)
 75 IF H THEN GO TO 90
 80 PRINT CHR$(C);
 85 GO TO 110
 90 LET X=C
100 GO SUB 320
110 PRINT " ";
170 NEXT A
180 STOP
300 REM DEC TO HEX
310 PRINT CHR$((X AND 28672) /4096+28);
    CHR$((X AND 3840)/256 + 28);
320 PRINT CHR$ ((X AND 240) /16+28);
    CHR$ (( X AND 15) +28);
330 RETURN
```


SPELLING QUIZ

PROGRAM DESCRIPTION

A spelling drill in two parts.

In the first part the teacher inputs five words to be used in the quiz.

In the second part the pupil is shown each word in turn and then asked to spell it. If incorrect the letters in error are highlighted. At the end the number of words correctly spelled is shown.

INSTRUCTIONS

Teacher: When the program has been loaded, type RUN NEWLINE. (Terminate all input by NEWLINE).

SPELLING QUIZ

INSTRUCTIONS continued

When prompted enter five words, each terminated by NEWLINE.

When the program has acknowledged these by 'THANKS', the program may be used by the pupil. (N.B. maximum total of 50 letters).

Pupil: Press NEWLINE CONT NEWLINE.

Five words will be shown in turn. After seeing the word, press NEW-LINE when prompted, and then repeat it. You will be shown any mis-takes you may make and at the end your score will be given.

The quiz may be rerun by the command GO TO 200.

LIST OF VARIABLES

M	=	total number of letters in all words + 1
Array C	=	character codes of letters in words
Array S	=	shows starting position of Array C where each word is held.
Array E	=	shows ending position of Array C where each word is held.
I	=	the sequence number of the words being entered/ tested.
W$	=	the word being input by the teacher
T	=	number of words spelled correctly
Z$	=	dummy string variable
X$	=	word as spelt by pupil
J	=	pointer to character in Array C. Also used as dummy — high value in one instance.
K	=	loop counter

Two subroutines used

SR500 — given a word W$ it enters its character codes at the next free position of Array C, setting S(I) and E(I) accordingly.

SR600 — prints a word from Array C, characters from and including position J in inverse video.

SPELLING QUIZ

SAMPLE RUN

ENTER o 5 o WORDS

DWARF)
DRAGON)
ELVES) Words set up by teacher
FAERIES)
LEPRECHAUNS)

Screen clears

THANKS

NEWLINE CONT NEWLINE entered

WORD 1 = DWARF

PRESS NL

Newline entered, screen clears

SPELL

DWARF

Screen clears

DWARF IS CORRECT

WORD 2 = DRAGON

PRESS NL

Newline entered, screen clears

SPELL

DRAGUN

Screen clears

DRAGUN IS WRONG – IT WAS

DRAGON

and so on

PROGRAM LISTING

```
20 LET M=1
30 DIM C(50)
40 DIM S(5)
50 DIM E(5)
70 PRINT "ENTER 5 WORDS"
80 FOR I=1 TO 5
90 INPUT W$
100 PRINT W$
110 GO SUB 500
120 NEXT I
130 CLS
140 PRINT "THANKS"
150 STOP
200 LET T=0
205 FOR I=1 TO 5
210 PRINT "WORD";I;"=",
215 LET J=99
220 GO SUB 600
225 PRINT "PRESS NL"
230 INPUT Z$
240 CLS
250 PRINT "SPELL"
260 INPUT X$

265 CLS
270 PRINT X$;" IS ";
280 FOR J=S(I) TO E(I)
290 IF X$="" THEN GO TO 360
300 IF NOT CODE (X$)=C(J) THEN GO TO 360
310 LET X$=TL$(X$)
315 NEXT J
317 IF NOT X$="" THEN GO TO 360
320 PRINT "CORRECT"
330 LET T=T+1
340 GO TO 380
360 PRINT "WRONG — IT WAS"
```

PROGRAM LISTING continued

```
370 GOSUB 600
380 PRINT
400 NEXT I
420 PRINT
430 PRINT "END. YOU GOT ";T;"RIGHT"
499 STOP
500 IF W$="" THEN STOP
505 IF M<51 THEN GO TO 510
507 PRINT "WORDS TOO LONG"
508 STOP
510 LET S(I)=M
520 LET C(M)=CODE(W$)
530 LET W$=TL$(W$)
540 LET E(I)=M
550 LET M=M+1
560 IF W$="" THEN RETURN
570 GO TO 520
600 FOR K=S(I) TO E(I)
602 IF K<J THEN GO TO 610
604 PRINT CHR$(C(K)+128);
606 GO TO 620
610 PRINT CHR$(C(K));
620 NEXT K
630 PRINT
640 RETURN
```

GRAPH PLOTTER

PROGRAM DESCRIPTION

Plots a graph of the equation $ax^2 + bx + c$ on positive labelled x and y axes. X axis is vertical and y horizontal.

20 lines used for x axis, 15 characters for y axis.

INSTRUCTIONS

Press RUN.

On invitation enter values for A, B and C and maximum x and y axis values, each terminated by NEWLINE.

If the resulting equations would be out of the specified axes range at any point, then the program reveals this and stops. Otherwise the graph is plotted by dots and labelled x and y axes.

N.B. If the graph is plotted generally close to the maximum value of the y axis then plotting may terminate prematurely. To remedy this re-run with a higher value for the y maximum.

LIST OF VARIABLES

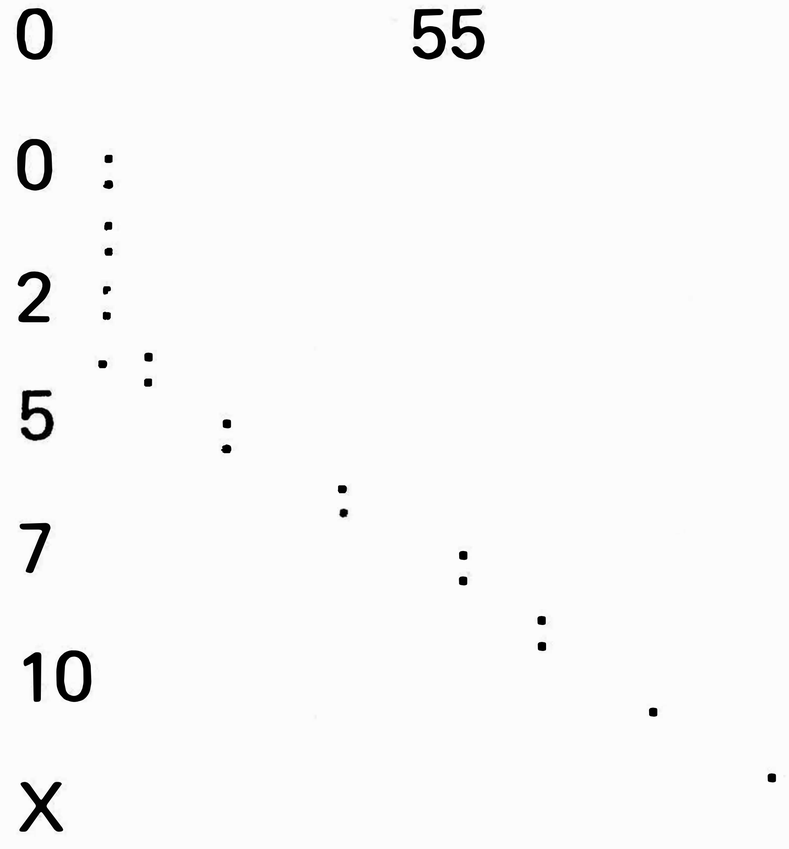
Array Y	=	scaled values of $y = ax^2 + bx + c$
Q	=	width in characters of graph
A,B, C	=	values in the equation $y = ax^2 + bx + c$
M	=	max.value on x-axis
N	=	max. value on y-axis
X	=	scaled counter for x-axis
E	=	true value on x-axis
Y	=	true value on y-axis

SAMPLE RUN

```
TO PLOT EQUATION AX2 + BX + C ENTER
A = 1   B = 0  C = 5
X-AXIS TO 10   Y-AXIS TO 110
Screen clears
```

GRAPH PLOTTER

SAMPLE RUN continued



and program terminates

PROGRAM LISTING

```
 10 DIM Y(20)
 15 LET Q=15
30 PRINT "TO PLOT EQUATION AX2+BX+C ENTER:"
 40 PRINT  "A=";
 50 INPUT A
 60 PRINT A;" B=";
 80 INPUT B
 90 PRINT B;" C=";
100 INPUT C
110 PRINT C
120 PRINT "X-AXIS TO ";
130 INPUT M
140 IF M>99 OR M<10 THEN GO TO 130
150 PRINT M;" Y-AXIS TO ";
160 INPUT N
170 IF N>999 OR N<10 THEN GO TO 160
180 PRINT N
190 FOR X=0 TO 20
200 LET E=X*M/20
210 LET Y(X)=(A*E*E+B*E+C)*Q/N
220 IF Y(X)<0 OR Y(X)>Q THEN GO TO 550
```

PROGRAM LISTING continued

```
230 NEXT X
240 CLS

250 PRINT " 0";
260 LET Y=N/2
270 GO SUB 600
320 LET Y=N
330 GO SUB 600
340 PRINT " Y";
350 FOR X=0 TO 20
360 IF X=(X/5)*5 THEN GO TO 420
380 PRINT " ";
390 GO TO 445
420 LET E=X*M/20
430 PRINT E;
440 IF E<10 THEN PRINT " ";
445 IF Y(X)<2 THEN GO TO 480
450 FOR I=1 TO Y(X)-1
460 PRINT " ";
470 NEXT I
480 PRINT "."
490 NEXT X
495 PRINT "X"
500 STOP
550 PRINT "OUT OF RANGE AT X=";E
560 STOP

600 PRINT "   ";
610 IF Y<100 THEN PRINT " ";
620 IF Y<10 THEN PRINT " ";
630 PRINT Y;
640 RETURN
```

HURKLE

PROGRAM DESCRIPTION

A 'Hurkle' is chosen to be at a random position in a 15 x 15 grid and the player has 5 goes at finding its position. After each guess he is told in which direction to move.

After five unsuccessful guesses the hurkle is revealed.

INSTRUCTIONS

Press RUN

Type NEWLINE when prompted

There is some delay between each grid display. X axis is horizontal and Y axis is vertical.

Input X and Y guesses when prompted, terminating each by NEWLINE.

Your position is shown by the go number.

At the end the hurkle's position is shown by H.

LIST OF VARIABLES

Z$	=	dummy variable
A	=	x co-ordinate of Hurkle
B	=	y co-ordinate of Hurkle
C	=	x co-ordinate guessed by player
D	=	y co-ordinate guessed by player
I	=	number of guess
Y	=	counter used to display grid

SAMPLE RUN

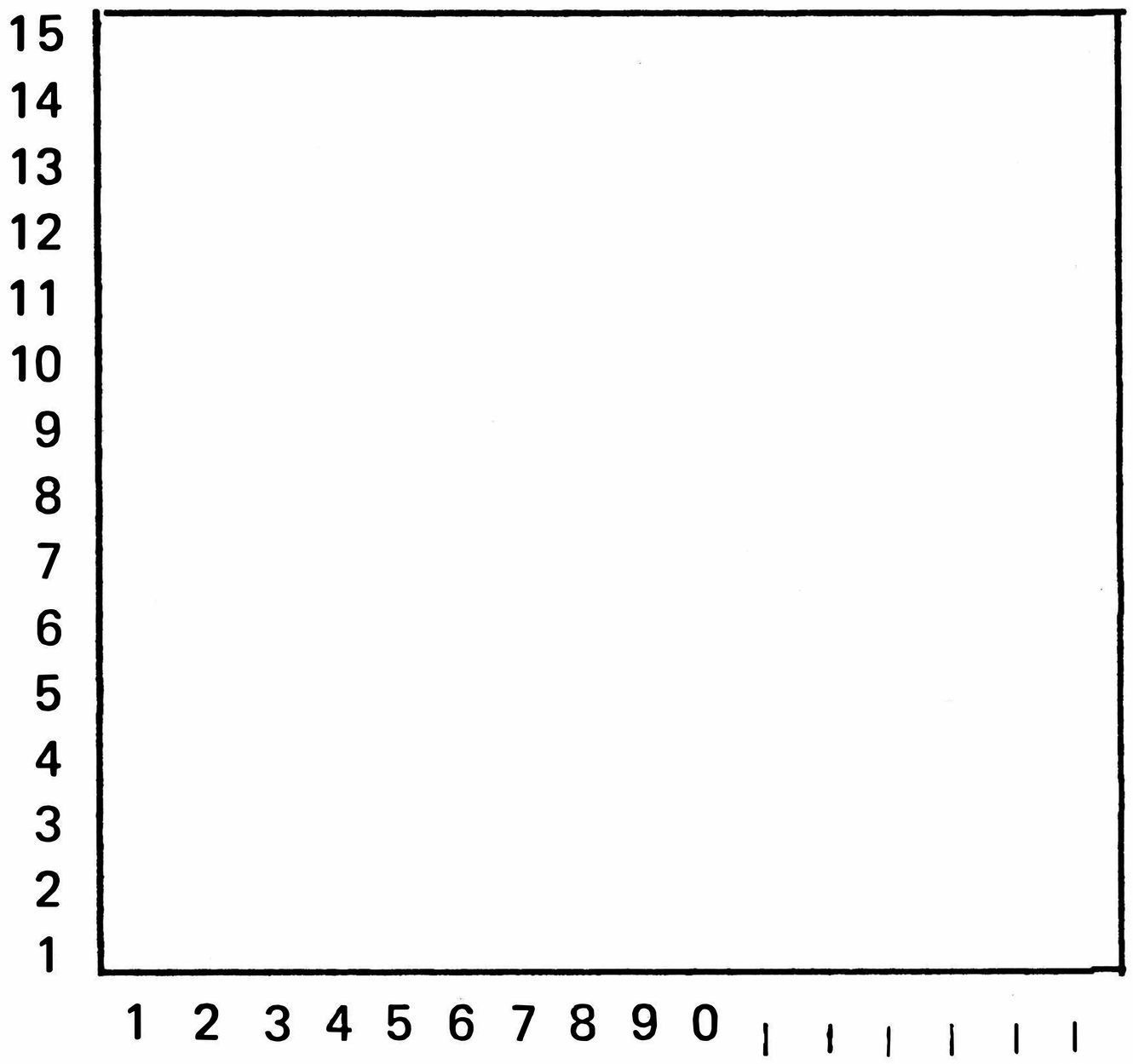
```
A HURKLE IS HIDING IN
A 15 x 15 GRID
YOU HAVE 5 TRIES TO FIND HIM
```

HURKLE

SAMPLE RUN continued

PRESS N/L

Newline entered, screen clears



PROGRAM LISTING

```
 2 REM HURKLE
 4 GO TO 48
 6 CLS
 8 LET Y=15
10 IF Y<10 THEN PRINT " ";
12 PRINT Y;"
14 LET Y=Y-1
16 IF Y>0 THEN GO TO 10
18 PRINT "  123456789111111"
20 PRINT ,"  012345"
22 PRINT
24 PRINT I;".X=";
26 INPUT C
28 PRINT C;" Y=";
```

SPACE

"

2 SPACES

3 SPACES

SPACE

PROGRAM LISTING continued

```
30 INPUT D
32 PRINT D;
34 IF C=A AND D=B THEN GO TO 72
36 POKE 17262+C-18*D,156+I
38 IF I=5 THEN RETURN
40 PRINT ".. ▒▒GO  ";CHR$(-51*(D<B)-56*(D>B));
   CHR$(-42*(C<A)-60*(C>A))
42 PRINT "PRESS N/L"
44 INPUT Z$
46 RETURN
48 RANDOMISE
50 PRINT "A HURKLE IS HIDING IN",,"A 15x15 GRID",,,
   "YOU HAVE 5 TRIES TO FIND HIM"
52 GO SUB 42
54 LET Z$=""
56 LET A=RND(15)
58 LET B=RND(15)
60 FOR I=1 TO 5
62 GO SUB 6
64 NEXT I
66 PRINT
68 PRINT "SORRY. IT WAS ";A;" ";B
70 GO TO 74
72 PRINT "  WELL DONE"
74 POKE 17262+A-18*B, 173
76 CLEAR
78 GO SUB 42
80 CLS
82 IF Z$ ="" THEN RUN
```

4-IN-A-LINE (2K)

Authors : R. Turner & C. Thornton

PROGRAM DESCRIPTION

This is a game for two players. The players move in turn. For his turn each player can 'drop' a counter into 1 of 7 columns. However the counters are subject to gravity and so fall to the lowest unoccupied square.

The winner is the first player to get 4-IN-A-LINE either horizontally, vertically or diagonally.

The computer checks for winning lines and displays a message if anyone wins (or if you try to cheat!)

INSTRUCTIONS FOR USE

The computer will display a 6 by 7 grid on the screen. It will also show at the bottom of the grid whose move it is and which symbol represents that player. If there are no empty squares and no one has won then the computer will declare it a draw.

LIST OF VARIABLES

A(42)	=	Positions in board. Contains code for character at that position.
C	=	Variable to store whose move it is.
Z	=	Your move. Also used in loop when printing board.
A,B	=	Used in loops when printing board.
D,E,G,I and K	=	Temporary variables to check if anyone has won.
A$	=	Response to question "play again?"

PROGRAM LISTING

```
 6 LET C=-1
 7 DIM A(42)
15 GO SUB 90
16 GO TO 300
17 PRINT "1 TO 7 PLAYER "; CHR$(39+C);" ";CHR$(19-C)
```

PROGRAM LISTING continued

```
 18 LET C=C=0 AND -1 OR C=-1 AND 0
 19 INPUT Z
 20 IF Z>7 OR Z<1 THEN GO TO 19
 23 GO SUB 170
 25 CLS
 30 GO TO 15
 90 PRINT, "4 IN A LINE"
 93 PRINT
 97 PRINT " 1  2  3  4  5  6  7"
100 FOR A= 1 TO 6
130 PRINT "  ▌  ▌  ▌  ▌  ▌  ▌ "
135 FOR Z=0 TO 5
140 PRINT " ";CHR$(A(42-(Z*6+A)));"▌";
145 NEXT Z
150 PRINT CHR$(A(42-(Z*6+A)))
155 FOR B=1 TO 6
156 PRINT "▬ ▬ ▬";CHR$(133);
157 NEXT B
158 PRINT "▬ ▬ ▬"
160 NEXT A
165 RETURN
170 FOR A=0 TO 41
180 IF A(A)=0 THEN GO TO 200
185 NEXT A
190 PRINT " DRAW "
195 GO TO 810
200 LET Z=7-Z
205 FOR A=0 TO 5
210 IF A((Z*6+A))=0 THEN GO TO 250
220 NEXT A
230 PRINT " FULL ";
235 INPUT Z
240 IF Z>7 OR Z<1 THEN GO TO 235
245 GO TO 200
250 LET A((Z*6+A))=20+C
260 RETURN
300 FOR A=0 TO 3
305 FOR B=0 TO 5
310 LET D=A(B+A*6)+A(A*6+B+6)+A(A*6+B+12)+A(A*6
    +B+18)
320 IF D=80 THEN GO TO 700
330 IF D=76 THEN GO TO 800
```

SHIF Q

NO SPACES BETWEEN SHIFT 'W'

PROGRAM LISTING continued

```
340 NEXT B
342 NEXT A
345 FOR A=0 TO 3
350 FOR B=0 TO 2
360 LET D=A(A*6+B)+A(A*6+B+7)+A(A*6+B+14)+
    A(A*6+B+21)
370 LET K=36+B–(A*6)
380 LET E=A(K)+A(K–5)+A(K–10)+A(K–15)
384 IF D=80 OR E=80 THEN GO TO 700
386 IF D=76 OR E=76 THEN GO TO 800
387 NEXT B
388 NEXT A
390 FOR B=0 TO 6
390 FOR F=0 TO 2
400 LET I=6*B
410 LET G=A(I+F)+A(I+F+1)+A(I+F+2)+A(I+F+3)
420 IF G=80 THEN GO TO 700
430 IF G=76 THEN GO TO 800
440 NEXT F
450 NEXT B
460 GO TO 17
700 PRINT " WELL DONE PLAYER A"
710 GO TO 810
800PRINT "WELL DONE PLAYER B"
810 PRINT "PRESS NEWLINE TO PLAY AGAIN"
820 CLEAR
830 INPUT A$
835 CLS
840 IF A$="" THEN GO TO 1
850 PRINT "BYE – THANKS FOR PLAYING"
```

REF. NO. G016 DOCUMENTATION

4-IN-A-LINE

SAMPLE RUN

Moves of Each Player

Player A	Player B
1	1
2	3
2	4
5	4
3	3
4	2
2	3

4 IN A LINE

1	2	3	4	5	6	7
	*	+				
	+	*				
+	*	+	*			
*	*	+	+	*		

(Alternates)
1 TO 7 PLAYER A*
1 TO 7 PLAYER B+

WELL DONE PLAYER A

PRESS NEWLINE TO PLAY AGAIN

THE MAZE GAME

Author : Mark Porter

PROGRAM DESCRIPTION

You are at the point X, You must reach the treasure (£) at the other corner by pressing the arrowed keys to move. At first glance it may seem easy, but as time goes on more and more of the blanks are filled in . . . .

INSTRUCTIONS

Press RUN.

For your move, enter a number from the range 5—9 or 0.

5,6,7,8	give movement in the arrowed direction
9	resets the X to the beginning — but the obstacle remains
0	If you can't move, or have got as close as you think you can, press this to obtain your score

any other number is ignored.

LIST OF VARIABLES

A	=	Array to hold maze element status values
B	=	Loop counter to set up array
C	=	Loop counter to print maze
D	=	User's move
E	=	Loop counter to update array
F	=	Random number in range 2—99 to pick element to update
X	=	Element number of user's position
Y	=	Buffer recording of X used to check validity of D

SAMPLE RUN

A regular pattern of black squares appears, with an X in the top left corner, and the treasure in the bottom right.

Enter 6 and press N/L.

The display blanks for a few seconds, and then reappears with the X

THE MAZE GAME

SAMPLE RUN continued

moved down by one square.

There's nothing to this, just pressing 6 and 8 to reach £ but after the fourth keystroke, two black squares appear to block your path.

Thereafter, more and more of the passageways are filled in, until either you reach the treasure or are trapped in the maze.

COMMENTS

Though the program listing only contains 39 lines, the array A and the graphic display take up a lot of the RAM space. Alterations to the program must therefore be balanced in order that error 4 is not encountered.

The loop E goes around 8 times, but each element must be picked twice for it to become blocked. This helps to maintain the number of squares blacking over at an approximately equal number in the middle of the game, and also makes it rather more difficult to retrace one's route after a couple of goes, though it usually remains open for a time. The function on line 205 is merely a subtraction from 18 of the number of moves to £, with like terms collected. In practice it is identical to the number of times the keys 6 and 8 have been used, less the number of times that 5 and 7 have been pressed, if 9 is not used.

PROGRAM LISTING

```
 10 RANDOMISE
 70 DIM A(99)
 80 FOR B=2 TO 99
 90 LET A(B)=2*((B/4)*4=B)+3
100 NEXT B
102 LET A(1)=4
104 LET X=1
110 FOR C=1 TO 99
120 IF A(C)=4 THEN PRINT "X";
130 IF A(C)=3 OR A(C)=2 THEN PRINT " ";
```

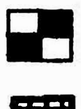
PROGRAM LISTING continued

```
150 IF A(C)=1 THEN PRINT CHR$(128);
160 IF C=99 THEN GO TO 185
170 IF (C/10)*10=C THEN PRINT
180 NEXT C
190 PRINT "ENTER MOVE:"
200 INPUT D
202 IF NOT D=0 THEN GO TO 210
205 PRINT "SCORE=";X+(X+10)/10+((100-X)/10)*10-92;"
    /18"
207 STOP
210 LET Y=X
220 LET Y=(1-Y)*(D=5) -(Y+10)*(D=6)-(Y-10)*(D=7)
    -(Y+1)*(D=8)-(D=9)
270 IF Y>100 OR Y<1 THEN GO TO 190
275 IF NOT Y=100 THEN GO TO 300
280 PRINT "YOU WIN. SCORE=18/18"
290 STOP
295 IF A(Y)=1 OR Y=X THEN GO TO 190
300LET A(Y)=4
310 LET A(X)=2
320 LET X=Y
330 FOR E=1 TO 8
340 LET F=RND(98)+1
345 IF F=X THEN LET F=F+1
350 LET A(F)=A(F)-1
360 IF A(F)<1 THEN LET A(F)=1
370 NEXT E
380 CLS
390 GO TO 110
```

185 PRINT "£"

APPENDIX ONE

COMPARISON OF Z80 INSTRUCTION CODES & ZX80 CHARACTER SET

DEC	HEX	OP Code	Character
0	00	NOP	space
1	01	LD BC,nn	null string
2	02	LD (BC),A	
3	03	INC BC	
4	04	INC B	
5	05	DEC B	
6	06	LD B,n	
7	07	RLCA	
8	08	EX AF,AF'	
9	09	ADD HL,BC	
10	0A	LD A,(BC)	
11	0B	DEC BC	
12	0C	INC C	£
13	0D	DEC C	$
14	0E	LD C,n	:
15	0F	RRCA	?
16	10	DJNZ disp	(
17	11	LD DE,nn	)
18	12	LD (DE),A	—
19	13	INC DE	+
20	14	INC D	*
21	15	DEC D	/
22	16	LD D,n	=
23	17	RLA	>
24	18	JR disp	<
25	19	ADD HL,DE	;
26	1A	LD A,(DE)	,
27	1B	DEC DE	.
28	1C	INC E	0
29	1D	DEC E	1
30	1E	LD E,n	2

DEC	HEX	OP Code	Character
31	1F	RRA	3
32	20	JR NZ,disp	4
33	21	LD HL,nn	5
34	22	LD(nn),HL	6
35	23	INC HL	7
36	24	INC H	8
37	25	DEC H	9
38	26	LD H,n	A
39	27	DAA	B
40	28	JR Z,disp	C
41	29	ADD HL,HL	D
42	2A	LD HL,(nn)	E
43	2B	DEC HL	F
44	2C	INC L	G
45	2D	DEC L	H
46	2E	LD L,n	I
47	2F	CPL	J
48	30	JR NC,disp	K
49	31	LD SP,nn	L
50	32	LD(nn),A	M
51	33	INC SP	N
52	34	INC(HL)	O
53	35	DEC(HL)	P
54	36	LD (HL),n	Q
55	37	SCF	R
56	38	JR C,disp	S
57	39	ADD HL,SP	T
58	3A	LD A,(nn)	U
59	3B	DEC SP	V
60	3C	INC A	W
61	3D	DEC A	X
62	3E	LD A,n	Y
63	3F	CCF	Z
64	40	LD B,B	?

N.B. n = single byte number nn = two byte number
disp= displacement (nn) = address

DEC	HEX	OP Code
65	41	LD B,C
66	42	LD B,D
67	43	LD B,E
68	44	LD B,H
69	45	LD B,L
70	46	LD B,(HL)
71	47	LD B,A
72	48	LD C,B
73	49	LD C,C
74	4A	LD C,D
75	4B	LD C,E
76	4C	LD C,H
77	4D	LD C,L
78	4E	LD C,(HL)
79	4F	LD C,A
80	50	LD D,B
81	51	LD D,C
82	52	LD D,D
83	53	LD D,E
84	54	LD D,H
85	55	LD D,L
86	56	LD D,(HL)
87	57	LD D,A
88	58	LD E,B
89	59	LD E,C
90	5A	LD E,D
91	5B	LD E,E
92	5C	LD E,H
93	5D	LD E,L
94	5E	LD E,(HL)
95	5F	LD E,A
96	60	LD H,B
97	61	LD H,C
98	62	LD H,D
99	63	LD H,E
100	64	LD H,H
101	65	LD H,L

DEC	HEX	OP Code	Character
101	65	LD H,L	?
102	66	LD H,(HL)	?
103	67	LD H,A	?
104	68	LD L,B	?
105	69	LD L,C	?
106	6A	LD L,D	?
107	6B	LD L,E	?
108	6C	LD L,H	?
109	6D	LD L,L	?
110	6E	LD L,(HL)	?
111	6F	LD L,A	?
112	70	LD (HL),B	?
113	71	LD (HL),C	?
114	72	LD(HL),D	?
115	73	LD(HL),E	?
116	74	LD(HL),H	?
117	75	LD(HL),L	?
118	76	HALT	?
119	77	LD(HL),A	?
120	78	LD A,B	?
121	79	LD A,C	?
122	7A	LD A,D	?
123	7B	LD A,E	?
124	7C	LD A,H	?
125	7D	LD A,L	?
126	7E	LD A,(HL)	?
127	7F	LD A,A	?
128	80	ADD A,B	
129	81	ADD A,C	inverse quote
130	82	ADD A,D	
131	83	ADD A,E	
132	84	ADD A,H	
133	85	ADD A,L	
134	86	ADD A,(HL)	
135	87	ADD A,A	
136	88	ADC A,B	
137	89	ADC A,C	

DEC	HEX	OP Code	Character
138	8A	ADC A,D	■
139	8B	ADC A,E	■
140	8C	ADC A,H	inverse £
141	8D	ADC A,L	inverse $
142	8E	ADC A,(HL)	inverse :
143	8F	ADC A,A	inverse ?
144	90	SUB B	inverse (
145	91	SUB C	inverse)
146	92	SUB D	inverse —
147	93	SUB E	inverse +
148	94	SUB H	inverse *
149	95	SUB L	inverse /
150	96	SUB (HL)	inverse =
151	97	SUB A	inverse >
152	98	SBC A,B	inverse <
153	99	SBC A,C	inverse ;
154	9A	SBC A,D	inverse ,
155	9B	SBC A,E	inverse .
156	9C	SBC A,H	inverse 0
157	9D	SBC A,L	inverse 1
158	9E	SBC A,(HL)	inverse 2
159	9F	SBC A,A	inverse 3
160	A0	AND B	inverse 4
161	A1	AND C	inverse 5
162	A2	AND D	inverse 6
163	A3	AND E	inverse 7
164	A4	AND H	inverse 8
165	A5	AND L	inverse 9
166	A6	AND (HL)	inverse A
167	A7	AND A	inverse B
168	A8	XOR B	inverse C
169	A9	XOR C	inverse D
170	AA	XOR D	inverse E
171	AB	XOR E	inverse F
172	AC	XOR H	inverse G
173	AD	XOR L	inverse H
174	AE	XOR (HL)	inverse I

DEC	HEX	OP Code	Character
175	AF	XOR A	inverse J
176	BO	OR B	inverse K
177	B1	OR C	inverse L
178	B2	OR D	inverse M
179	B3	OR E	inverse N
180	B4	OR H	inverse O
181	B5	OR L	inverse P
182	B6	OR (HL)	inverse Q
183	B7	OR A	inverse R
184	B8	CP B	inverse S
185	B9	CP C	inverse T
186	BA	CP D	inverse U
187	BB	CP E	inverse V
188	BC	CP H	inverse W
189	BD	CP L	inverse X
190	BE	CP (HL)	inverse Y
191	BF	CP (A)	inverse Z
192	CO	RET NZ	?
193	C1	POP BC	?
194	C2	JP NZ,nn	?
195	C3	JP nn	?
196	C4	CALL NZ,nn	?
197	C5	PUSH BC	?
198	C6	ADD A,n	?
199	C7	RST O	?
200	C8	RET Z	?
201	C9	RET	?
202	CA	JP Z,nn	?
203	CB	two byte instruction code	?
204	CC	CALL Z, nn	?
205	CD	CALL nn	?
206	CE	ADC A,n	?
207	CF	RST 8	?
208	DO	RET NC	?
209	D1	POP DE	?
210	D2	JP NC,nn	?
211	D3	OUT port,A	?

DEC	HEX	OP Code	Character
212	D4	CALL NC,nn	"
213	D5	PUSH DE	THEN
214	D6	SUB n	TO
215	D7	RST 10H	;
216	D8	RET C	,
217	D9	EXX	)
218	DA	JP C,nn	(
219	DB	IN A,port	NOT
220	DC	CALL C,nn	—
221	DD	two byte instruction code	+
222	DE	SBC A,n	*
223	DF	RST 18H	/
224	EO	RET PO	AND
225	E1	POP HL	OR
226	E2	JP PO,nn	**
227	E3	EX (SP),HL	=
228	E4	CALL PO,nn	>
229	E5	PUSH HL	<
230	E6	AND n	LIST
231	E7	RST 20H	RETURN
232	E8	RET PE	CLS
233	E9	JP (HL)	DIM
234	EA	JP · PE,nn	SAVE
235	EB	EX DE,HL	FOR
236	EC	CALL PE,nn	GO TO
237	ED	two byte instruction code	POKE
238	EE	XOR n	INPUT
239	EF	RST 28H	RANDOMISE
240	FO	RET P	LET
241	F1	POP AF	?
242	F2	JP P,nn	?
243	F3	DI	NEXT
244	F4	CALL P,nn	PRINT
245	F5	PUSH AF	?
246	F6	OR n	NEW
247	F7	RST 30H	RUN
248	F8	RET M	STOP

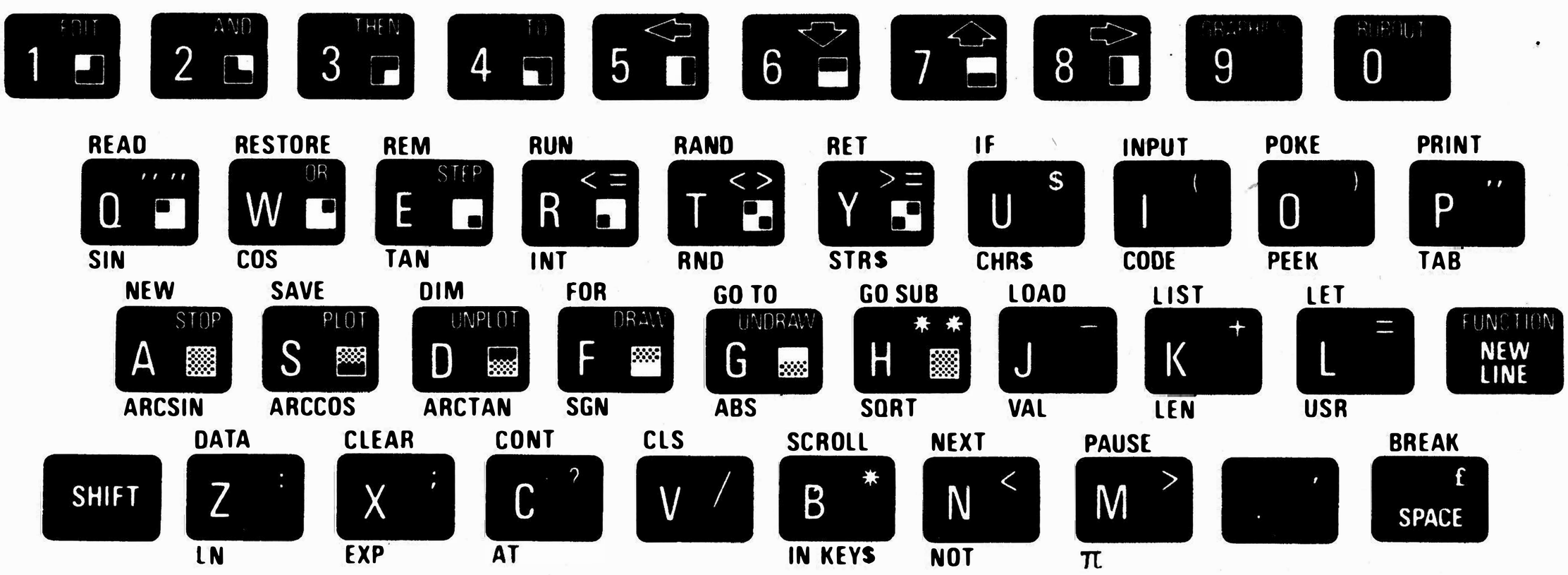
DEC	HEX	OP Code	Character
249	F9	LD SP,HL	CONTINUE
250	FA	JP M,nn	IF
251	FB	EI	GO SUB
252	FC	CALL M,nn	LOAD
253	FD	two byte instruction code	CLEAR
254	FE	CP n	REM
255	FF	RST 38H	?

APPENDIX TWO – THE 8K ROM

A2.1 INTRODUCTION

The long-awaited 8K BASIC ROM for the ZX80 was launched at the Third Personal Computer World Show, 4 - 6 September 1980 in London. This coincided with the time at which this edition was going to print and the following section is based upon initial use of a prototype. Material is reproduced from the Sinclair specification sheet with permission of Science of Cambridge.

The new ROM is sold as a plug-in replacement for the 4KROM described in this volume. It comes with a new keyboard template to provide for the new BASIC functions and instructions. This looks like:



Sinclair ZX80 8K BASIC

A.2.2 NEW FEATURES

Data Storage

Numbers are stored in five bytes in floating point binary form giving a range of numbers

$$9 \times 10^{-44} \leqslant n \leqslant 1.1 \times 10^{43}$$

Numbers cannot be defined as integers, but storage of integers between 0 and 255 can be achieved by the "fiddle" of using character codes of

string-defined variables.

String arrays are available and must be dimensioned. Substrings are obtained by the keyword TO, e.g.

```
99 LET B$ = A$ (2 TO 4)
```

Undimensioned strings are still available but these may now be concatenated using the + operator.

Instructions

READ, DATA and RESTORE statements are available in their standard form.

Graph plotting is achieved by several new instructions. Each uses pixels (quarter character squares).

1) PLOT x,y blacks in a pixel at coordinate (x,y). The screen consists of 64 positions in the x—direction and 44 in the y—direction; both increment in the normal graph directions.
2) UNPLOT x,y blanks out a pixel at (x,y)
3) DRAW a,b draws a line from the current PLOT position, say (x,y), to (x+a, y+b).
4) UNDRAW a,b — reverse of DRAW.

A GRAPHICS key is included to allow **ALL** characters and their inverses to be entered from the keyboard.

Named cassette program files are now available using

SAVE filename and LOAD filename

The authors cannot comment on its usefulness as this has not been assessed.

A STEP feature is now included in FOR statements

e.g. FOR I=1 TO 20 STEP 4

Different PRINT options are specified so that the user can print at a given position on the screen

e.g. PRINT AT 10,20

or print to a given number of digits or in E format

e.g. PRINT TO 4

The system variable RAMTOP may be altered using NEW n, in order to protect machine code programs stored above it (see Section 4.7).

Finally, moving display features are included. The statement SCROLL causes the display file to SCROLL up one line. Also, the piece de resistance, the PAUSE n instruction sends the display file to the screen for n frames or until a key is pressed.

Functions

A function key is included above NEWLINE. The table below shows new functions only.

FUNCTION	OPERAND	MEANING
ARCOS	number	standard trigonometrical functions (using radians)
ARCSIN	"	
ARCTAN	"	
COS	"	
SIN	"	
TAN	"	
EXP	"	e^x
INKEY$	character	scans keyboard and obtains character of key depressed (if any)
INT	number	integer
LN	number	natural logarithm
PI	—	TT (=3.1415927)
SGN	number	sign (−1, 0 or +1)
SQRT	number	square root

Operators

Combinations of relational operators are permitted, e.g..

<= , >= and < > (not equal to)

A2.3 SAMPLE PROGRAM

The new owner will probably be most interested in seeing a demonstration of the moving display. The following simple program demonstrates some of these features :

```
10 PRINT "MOVING DISPLAY"
20 PRINT "=============="
30 PRINT "DIRECTION OF MOVEMENT:"
40 PRINT "I. NORTH"
50 PRINT "2. EAST"
60 PRINT "3, NORTH-EAST"
70 PRINT "INPUT 1,2 OR 3"
80 INPUT N
90 LET N=INT(N)
100 IF N>3 OR N<1 THEN GO TO 80
110 PRINT N
120 IF N=1 THEN GO TO 180
130 PRINT "ENTER TIME BETWEEN MOVES"
140 PRINT "IN TENTHS OF SECONDS"
150 INPUT T
160 LET T=INT(T)
170 IF T<5 OR T>100 THEN GO TO 150
180 CLS
190 PLOT 1,1
199 GO TO 50*N+150
200 FOR I=1 TO 15
210 SCROLL
220 NEXT I
249 STOP
250 FOR I=2 TO 40
260 UNPLOT I-1,1
270 PLOT I,1
280 PAUSE T*5
290 NEXT I
299 STOP
300 FOR I=2 TO 40
310 UNPLOT I-1,I-1
320 PLOT I,I
330 PAUSE T*5
340 NEXT I
```

INDEX

ABS 27, 32
Arrays Sect. 3.5, 42
Arithmetic subroutines Sect. 4.6
Assembly 81
Assignments & Calculations Sect. 3.2
Bits 15, 16
Books 44
Bytes 15
Cassette interface 86
Cassette recorder 7, 8, 9
Character printing routine 69
Characters Sect. 3.6, 84
Characters table 54
CHR$ 27,52
CLEAR 25
CLS 26
CLS routine 69
CODE 27
Commands 24
Conditions Sect. 3.4, 86
Connections 7
Construction & hardware — Chapter 5
CONTINUE 12, 25
Current display position 67
Current position store routine 69
Cursor 24
Cursor table 56
Debugging 20
DIM 26, 38
Display file 23, 63
Documentation 21
Edge connector 87
EDIT 23
End of program sequence 70
Expressions 28
Flowcharts 18, 19
FOR statement 37
Four-in-a-line 108
Functions 26, 31, 40
GO SUB 26, 35
GO TO 25, 34
Graph plotter 102
Hurkle 105
IC's 90
IF statement 26, 35
Input II
INPUT statement 25, 32
Input device 14
Instructions 25
Integers 32
Interpreter 17
Interrupting programs 12
Inverse printing 47
Jumps Sect. 3.4
Keyboard 86
Keywords 10, 24
Languages 17
LET 25, 28
Lines 23
LIST 24
LOAD 24
LOAD routine 62
Loading from cassette Sect. 1.3
Loading from listings Sect. 1.4
Logical operators 30
Logical values 29
Machine Code Sect. 3.7, Chap. 4
Main memory 15
Masking 30
Maze Game 112
Memory display 95
Modes II
Modules 18
Monitor Chap. 4
Moving display routine Sect. 4.8
Moving RAMTOP down 77
NEW 25
New ROM — Appendix 2
Number printing routine 72
Operators 37, 36
Output device 14
Pattern recognition 91

INDEX cont'd

PCB 82
PEEK 27, 44, 51
POKE 26, 44
Power supply 9, 82, 87
PRINT 25, 32
PRINT routines 63
Processor 14, 15
RAM 23, 46, 89
RANDOMISE 26
Relational operators 40
REM 26
Rerunning programs 12
RETURN 26
RND 26, 31
RND routine 74
ROM 43, 46, Chap. 4, 88
Routines 33, Chap. 4
RUBOUT 11, 23
RUN 25
Running programs Sect. 1.5
SAVE 24
SAVE routine 61
Screen & keyboard routine 59
Secondary memory 16, 17
Special characters 53
Spelling quiz 97
Stack 77
STR$ 27
Symbols 20, 24, 28
System variables 45, 46
Tables 53
Television 7, 8, 84
Testing programs 20
Timing 47
TL$ 27
USR 27, 45, 57
Universal M/C loader 49
–X, Y Convention 68
Z–80 microprocessor 4, 83
Z–80 opcodes – Appendix 1

Printed by Prontaprint · Middlesbrough · Darlington